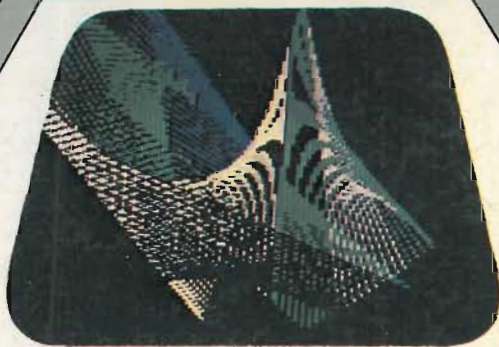
INTRODUZIONE AL
BASIC

EDIZIONE ITALIANA

PIERRE LE BEUX

GRUPPO EDITORIALE JACKSON

INTRODUZIONE AL BASIC

di
Pierre
Le Beux

GRUPPO
EDITORIALE
JACKSON
Via Rosellini, 12
20124 Milano

SOMMARIO

APPENDICI

PREMESSA

Questa opera è indirizzata sia alla schiera di lettori che si accostano per la prima volta all'informatica, sia a coloro che hanno già esperienza dei linguaggi di programmazione.

Per la prima categoria di lettori non è necessaria nessuna conoscenza preliminare, per cui si consiglia la lettura passo per passo per acquisire i concetti di base necessari alla comprensione dei capitoli seguenti. Per la seconda categoria di lettori, l'approccio al libro deve essere differente: essi potranno direttamente incominciare la lettura al secondo capitolo per conoscere gli elementi di base del linguaggio e, in seguito, interessarsi alla sua particolarità e provare a programmare direttamente in 'BASIC' gli esercizi che sono proposti.

Gli ultimi capitoli sono più specialistici: saranno quindi interessanti per coloro che vogliono conoscere le estensioni standard del linguaggio.

Riassumendo, il capitolo 1 è una introduzione al vocabolario e ai concetti di base dell'informatica.

Il capitolo 2 mostra le generalità del linguaggio BASIC.

Il capitolo 3 descrive le istruzioni fondamentali del linguaggio, con l'ausilio di esercizi.

Il capitolo 4 presenta concetti più elaborati del linguaggio di base, precisamente il trattamento degli elenchi, dei prospetti, delle matrici, la nozione di funzione e di sottoprogrami.

Nel capitolo 5 vengono considerati problemi di trattamento di schedari, che, benchè siano propri di ciascun sistema, sono presentati in modo sufficientemente generale, così da permettere di adattarsi a qualunque sistema.

Nel capitolo 6 sono presentati i procedimenti grafici e le possibilità offerte, per cui alcuni sistemi studiati per programmi tipo possono essere facilmente adattati ad ogni tipo di sistema.

Infine, il capitolo 7 comprende un certo numero di facilitazioni offerte dai sistemi BASIC specialmente su microelaboratori.

Speriamo che così il lettore tragga dall'opera le risposte a tutte le questioni che egli può porsi sul linguaggio BASIC o sulle estensioni.

PREFAZIONE

Il BASIC è un linguaggio di programmazione ad alto livello. Il suo nome è tratto da «Beginner's All purpose Symbolic Instruction Code» ed è stato sviluppato inizialmente al «Darmouth College».

È un linguaggio interattivo e interpretato, cioè ogni istruzione è tradotta in linguaggio-macchina ed eseguita immediatamente appena viene riconosciuta come corretta. Se ci sono errori, questi sono subito rilevati in maniera tale da poterli correggere. Appunto per ciò il BASIC è un linguaggio facile da apprendere e da utilizzare.

Il suo carattere interattivo facilita l'accesso all'elaboratore, senza che sia necessario familiarizzarsi con le idiosincrasie dei sistemi di sfruttamento degli elaboratori: carte di riscontro per il linguaggio astruse per i principianti, segnali d'errore spesso poco espliciti, momenti di disattenzione...

Si deve notare, tuttavia, che questo linguaggio può essere ugualmente utilizzato da professionisti che vogliono verificare rapidamente un algoritmo, trattare un problema di matematica o di statistica elementare o comunque consultare una piccola biblioteca personale di dati. BASIC è il linguaggio che più soventemente è utilizzato su macchine che lavorano in tempo parziale. Lo sviluppo della tecnologia dei micro-elaboratori e dei sistemi personali ha ridato al linguaggio BASIC l'importanza dovuta essenzialmente alla sua facilità di apprendimento e al suo carattere interattivo. È quindi evidente la ragione per cui abbiamo pensato che fosse utile scrivere su ciò un'opera di informazione di base e, in particolare, di presentare le versioni attuali disponibili sui differenti tipi di microelaboratori.

Resta da dire che esistono numerosi detrattori del linguaggio: è obsoleto e non permette di strutturare bene i programmi, il suo carattere interattivo può dare cattive abitudini di programmazione. D'altra parte, i programmi scritti in BASIC vengono eseguiti più lentamente degli altri ed utilizzano come lingua base l'Inglese, cosa che lo rende meno adatto all'insegnamento nei paesi non anglosassoni... Per quanto riguarda le prime critiche, esse sono in buona parte non giustificate: esistono linguaggi più moderni, ma sono di più difficile interpretazione (se si esclude APL). Il suo carattere interattivo può dare programmi mal costruiti, ma questi possono essere considerati come dei bro-

gliacci da buttare e che possono eventualmente essere utilizzati come base per riprogrammare in maniera più utile con altri linguaggi.

Per quanto concerne le ultime critiche, esse sono soltanto parzialmente valide. La velocità di esecuzione non ha più senso, nel momento in cui l'utilizzatore ottiene una risposta quasi istantanea. In effetti, nella maggior parte dei casi, un utente ottiene la risposta più rapida con un microelaboratore piuttosto che con un computer che lavora a tempo parziale o di cui non è il solo utente. Non parliamo poi dei trattamenti per parti, che daranno ritardi ancora superiori.

Il carattere 'anglofono' del linguaggio è secondario nella misura in cui si può, ed è quello che cercheremo di fare, scrivere in un BASIC italiano, con programmi di traduzione appropriati e facili da impostare.

Detto ciò, il BASIC resta un linguaggio pratico, semplice e disponibile attualmente su tutti i micro-elaboratorati del mercato. Proprio per questo il numero dei programmi attualmente scritti secondo questo linguaggio aumenta in maniera apprezzabile di giorno in giorno!

CAPITOLO 1

INTRODUZIONE

Il linguaggio BASIC è un linguaggio di programmazione evoluta, definito ad alto livello, poichè la sua definizione è indipendente dall'elaboratore o dalla macchina sulla quale i programmi scritti in questo linguaggio saranno eseguiti. Prima di studiare il linguaggio propriamente detto, è dunque necessario presentare la struttura di base degli elaboratori attuali e i concetti di base della programmazione.

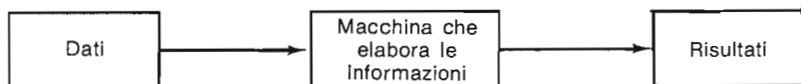
In questo capitolo, dunque, verranno presentate successivamente generalità su come trattare le informazioni, sulla struttura e il funzionamento degli elaboratori, sugli algoritmi e sugli schemi a blocchi. Queste nozioni non sono proprie del BASIC, ma costituiscono un preliminare necessario per chi sta imparando o per chi sarà in futuro l'utente di questo linguaggio. Il lettore che già conosce queste nozioni può dunque passare direttamente ai capitoli successivi, nei quali verrà presentato il linguaggio propriamente detto.

1. GENERALITÀ SUL TRATTAMENTO AUTOMATICO DELL'INFORMAZIONE

Lo scopo della programmazione è permettere di specificare ad una macchina un certo lavoro da svolgere in maniera automatica.

Per fare ciò, bisogna generalmente fornire alla macchina i valori di certi parametri che si chiamano *dati.* Successivamente la macchina effettuerà su questi dati un certo numero di operazioni, seguendo un certo schema di funzionamento che le sarà stato precisato, sia una volta per tutte, sia ad ogni singola domanda (questo schema di funzionamento corrisponde a quello definito 'programma').

Infine, è evidente che tutto ciò presenta interesse, in quanto la macchina fornisce un certo numero di *risultati.*

Così schematicamente si ha:

In questo schema l'uomo interviene solamente per alimentare la macchina (con i dati) e raccogliere i risultati (Naturalmente egli interviene pure per elaborare il programma).

1-1. Macchine a programma prestabilito

È importante sottolineare che il trattamento automatico dell'informazione può essere effettuato attraverso connessioni elettriche, meccaniche, ottiche, ... in funzione dei differenti elementi costituenti la macchina. Si dice allora che il programma è prestabilito (nel caso di una macchina elettronica si dirà cablato). Le macchine a programma prestabilito sono obbligatoriamente specializzate per un certo tipo di programma automatico (si può dire in questo senso che la calcolatrice di Pascal fu la prima macchina programmata meccanicamente per effettuare operazioni aritmetiche). Tra le macchine a programma cablato, è possibile citare tutti gli automatismi elettronici, dall'ascensore o dal cancello automatico, fino ai vari apparecchi automatici utilizzati nell'industria (macchine attrezzate, catene di montaggio, ecc.).

La caratteristica principale delle macchine a programma prestabilito è che il programma è codificato in maniera definita e irreversibile nella struttura stessa della macchina.

1-2. Macchine a programma registrato

Per permettere una maggiore flessibilità, si fa ricorso a macchine a *programma registrato.* Il primo tipo di macchine a programma registrato fu progettato da Babbage (matematico inglese del XIX secolo), con l'aiuto di un programma esterno alla macchina e codificato su un supporto continuo. Su questo supporto sono dunque registrate le differenti istruzioni costituenti il programma.

Il vantaggio di questo tipo di macchina sta nel fatto che si può cambiare facilmente il programma, e, di conseguenza, la macchina può trattare differenti tipi di problemi: è sufficiente scrivere nuovamente il programma sul supporto. Si dice allora che il programma è 'universale'.

Così, paradossalmente, il concetto di macchina universale non è obbligatoriamente sinonimo di complessità: la macchina di Türing (matematico in-

glese del XX secolo), composta da una testa di lettura e da un nastro contenente il programma che può avanzare o arretrare davanti alla testa, è capace di eseguire tutte le operazioni e tutti gli algoritmi eseguibili sulle macchine più moderne.

Il solo svantaggio di tale tipo di macchina è che la programmazione *è* lunga e fastidiosa, e che, in più, il tempo di esecuzione per problemi complessi può essere complessivamente lungo.

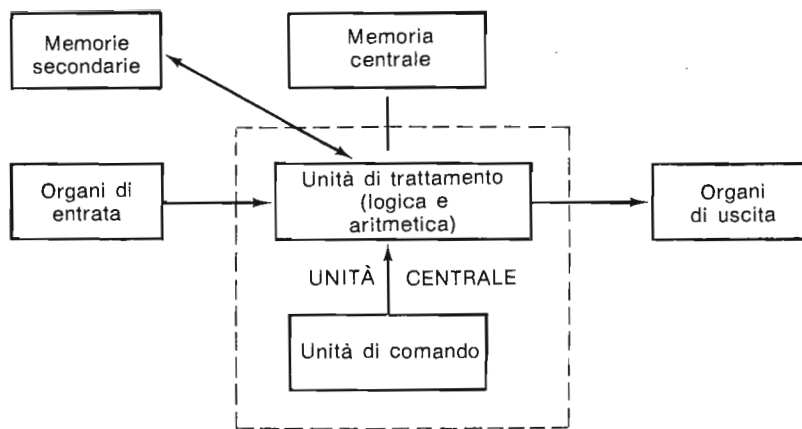
C'è stato bisogno del genio del matematico Von Neumann per realizzare (1946), in qualche modo, la sintesi tra le macchine a programma prefissato e quelle a programma registrato esterno: l'idea fondamentale è stata di considerare i programmi come dei dati che potevano essere immagazzinati internamente alla macchina (in una memoria interna, nella quale si possono registrare differenti informazioni che sono sia istruzioni di programma che dati).

Quest'idea e questa concezione sono alla base delle moderne macchine o elaboratori. Il passo successivo alla invenzione degli elaboratori è che la programmazione è divenuta una disciplina importante e che l'informatica è divenuta una scienza ed una tecnica, ben distinta dalle matematiche e dall'elettronica.

Nota. — Nelle attuali macchine, esiste una nuova concezione di programma prestabilito, in particolare quando si parla di 'microprogrammazione'. Di fatto si tratta di programmi registrati in memorie dette *morte* che non si possono modificare (ROM: Read Only Memory) o che sono soltanto raramente modificabili. Queste tecniche permettono di costruire una macchina relativamente complessa, partendo da una micromacchina avente istruzioni estremamente elementari che servono alla scrittura di microprogrammi, con i quali realizzare delle funzioni o delle operazioni più complesse. Su questo tipo di programmazione non ci soffermeremo qui.

2. STRUTTURA E FUNZIONAMENTO DI UN ELABORATORE

Non si tratta qui di dare una spiegazione dettagliata di un elaboratore, ma è importante conoscere lo strumento da utilizzare: da ciò deriva la necessità di presentare i differenti organi che costituiscono una macchina per elaborare informazioni.

Gli elaboratori sono anche chiamati 'calcolatori digitali', per differenziarli dai calcolatori analogici. Un calcolatore digitale lavora su dati discreti (non continui), codificati da numeri binari (sistema di numerazione che utilizza la base 2).

Schema di un elaboratore

In questo schema si distinguono i seguenti organi:

2-1. La memoria centrale

È in questa memoria che sono contenuti programmi e dati. Dal punto di vista tecnologico, una memoria principale è caratterizzata dai seguenti elementi:

- la *tecnologia* utilizzava prime strutture toroidali magnetiche, ora semiconduttori: MOS (Metal Oxide Semiconductor), bipolare (memorie a bolle magnetiche, ecc.);
- Il *ciclo di memoria di base*, che indica i tempi necessari per leggere o scrivere una certa quantità di informazioni in memoria. Nelle macchine attuali, il ciclo di base è dell'ordine dei microsecondi, μs (i più rapidi sono dell'ordine di qualche centinaio di nanosecondi).

Organizzazione della memoria

La più piccola informazione memorizzabile si dice bit (contrazione di *bi*nary digi*t*) e rappresenta una cifra binaria (0 o 1).

La capacità di una memoria potrebbe essere numerata col numero totale di bits, ma, per facilitare il trattamento e l'indirizzamento dei dati contenuti nella memoria, essa è divisa in celle elementari di più bits che si chiamano *parole memoria* («*bytes*») che sono accessibili in blocco da un indirizzo che rappresenta la posizione della cella nella memoria. (Per ragioni di omogeneità tutte le celle hanno le stesse dimensioni). Una memoria è anch'essa caratterizzata dalla dimensione dei suoi bytes: 8 bits, 16 bits, 24 bits, 32 bits, ... Per

quanto questa dimensione vari a seconda delle macchine e dei loro costruttori, soventemente si verifica che le dimensioni delle parole sono multipli delle parole di 8 bits (chiamati *ottetti*). Per le piccole macchine (microelaboratori) la parola è con più frequenza di 8 bits.

Per i mini elaboratori la parola di 16 bits è la più frequentemente utilizzata. Per le macchine più potenti è di 32 bits o più. Bisogna tuttavia notare che le frontiere tra micro, mini e grossi elaboratori tendono ad abbattersi rapidamente.

Grandezza di una memoria centrale

Infine, un altro parametro caratterizzante la memoria centrale è il numero totale di parole contenute nelle memorie. La grandezza totale si esprime come multipli di K = 1024 bytes ($1024 = 2^{10}$). Così, si parlerà di una memoria di 4K, di 16K, di 32K... Questo parametro è pure utilizzato per indicare lo spazio occupato da un programma: così, se si ha un programma che occupa 20K, questo non potrà essere eseguito da una macchina avente una memoria centrale di 16K.

2-2. L'unità di trattamento

Si compone di due blocchi funzionali.

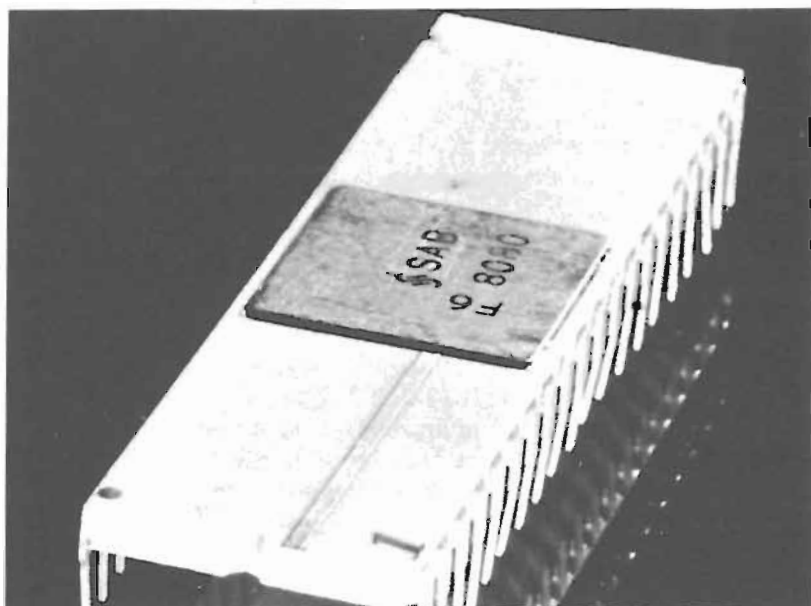
Attualmente le unità centrali possono essere integrate su un solo circuito ad alta integrazione, detto microprocessore (LSI).

L'unità di trattamento costituisce la parte attiva della macchina, la parte che elabora i dati per fornire i risultati. In questa unità di trattamento, si avranno sempre una unità aritmetica (addizione, sottrazione, moltiplicazione ecc..) e una unità logica (confronto tra operazioni logiche). Quest'ultima unità dispone poi di memorie specializzate, nelle quali si effettuano le operazioni, e che si chiamano registri (accumulatori, registri di indice, pile, ecc...).

L'unità di trattamento (CPU, dall'inglese Central Processing Unit) è anche caratterizzata da un repertorio di *istruzioni*, utilizzato per indicare il lavoro da eseguire (ma può anche essere utilizzata per ordinare dei dati, da cui il nome di ordinatore). Queste istruzioni sono elementari (unire due parole oppure compararle) e sono le sole ad essere realmente eseguite dalla macchina.

È importante sottolineare che, per le unità di trattamento, le istruzioni possono essere considerate dei dati che sono decodificati e che generano un certo numero di comandi e di operazioni all'interno dell'unità centrale.

L'unità di comando

È quell'organo che decodifica le istruzioni e le trasmette all'unità di trattamento. Questo organo è in relazione con la memoria principale (per leggere o

Un microprocessore

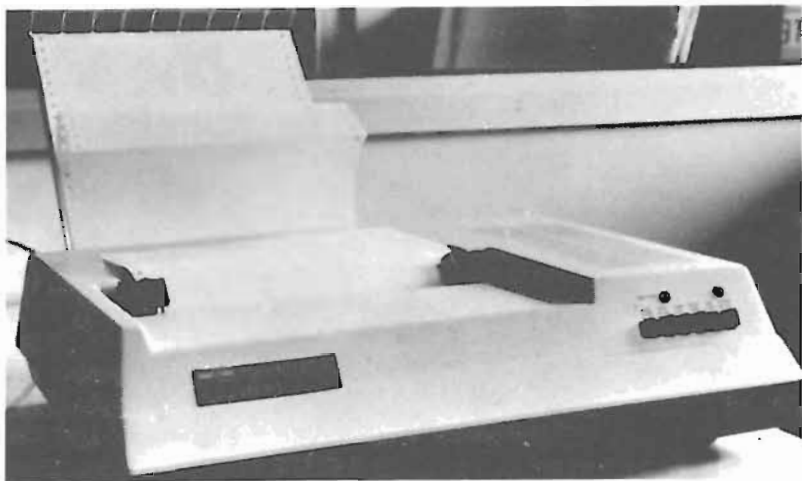
scrivere in memoria delle informazioni) e con l'unità di lavoro che effettua le operazioni propriamente dette.

Gli organi di comando servono altresì a decodificare gli ordini provenienti dall'esterno o diretti verso l'esterno (organi di entrata - uscita).

L'unità di comando può essere realizzata con l'aiuto di logiche elettroniche, o può essa stessa funzionare, partendo da una memoria morta contenente delle microistruzioni. Si dice allora che l'unità centrale è microprogrammata.

2-3. Gli organi di entrata-uscita

Questi organi servono a far comunicare la macchina con la realtà esterna e particolarmente a fornire alla macchina i programmi e i dati che si vuole vengano eseguiti, per ottenere, di ritorno, dei risultati. Questi organi sono quelli che permettono all'uomo di comandare e utilizzare la macchina.

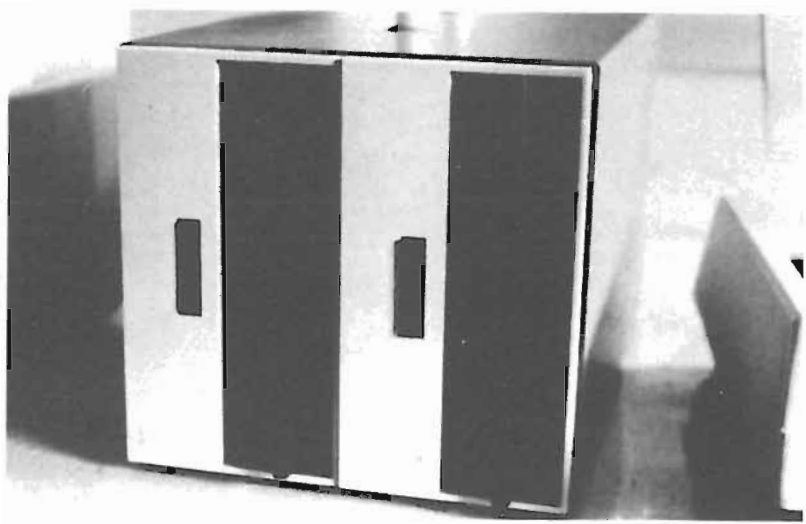
Gli organi di entrata-uscita sono anche detti *organi periferici* dell'elaboratore. Si possono citare come esempi: i lettori di schede e nastri perforati, i perforatori degli stessi, le stampanti, le telescriventi, le traccianti o plotters, le unità di visualizzazione ecc.

Una stampante

In questa categoria bisogna ugualmente inserire le *memorie secondarie*, su cui si possono leggere o scrivere informazioni che si vogliono conservare a lungo. In effetti, a causa del suo carattere universale e limitato, è escluso che la memoria principale di un elaboratore venga utilizzata per l'immagazzinaggio di informazioni (programmi o dati) più a lungo di quanto necessiti per l'esecuzione di un programma. È dunque possibile utilizzare delle memorie dette secondarie, dove si potranno immagazzinare le informazioni in modo permanente o quasi.

Gli organi delle memorie secondarie sono di due tipi:

- le memorie ad accesso *sequenziale* (nastri magnetici o perforati, schede, cassette)
- le memorie secondarie ad accesso *aleatorio* (dischi magnetici flessibili o hard disk).

Associati a questi differenti tipi di memorie secondarie, si hanno degli organi periferici corrispondenti (l'avvolgitore dei nastri, delle cassette, e degli organi che controllano queste operazioni).

Nota. — Un certo numero di programmi possono essere immagazzinati in permanenza in una parte della memoria centrale costituita da memorie morte (ROM).

Una unità di dischi flessibili

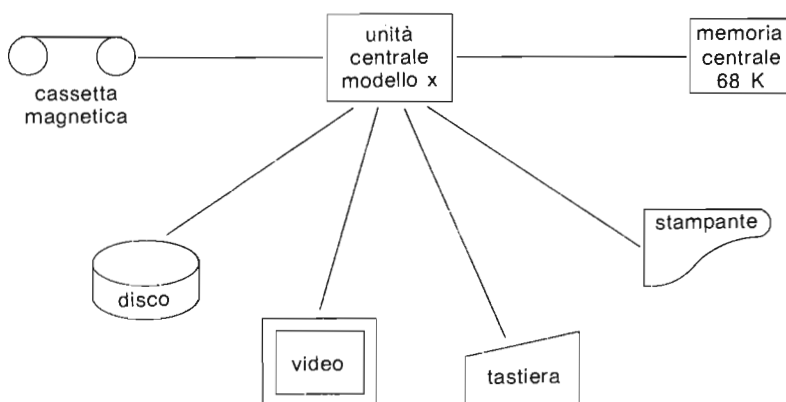
In alcuni casi è necessario considerare queste memorie secondarie come una estensione della memoria principale. Utilizzando tecniche più sofisticate (overlays, swapping, segmentation, pagination), si può dare all'utente l'impressione che a sua disposizione esista una memoria quasi illimitata. Non staremo a dare dettagli in questa sede su queste tecniche dette di *memoria virtuale,* ma è importante sapere che esse esistono e che sono sempre più in espansione.

C'è da tener presente poi che gli organi di entrata-uscita sono generalmente più lenti che l'unità centrale (unità di lavoro + organi di comando) e ciò comporta il crearsi di strozzature che si producono frequentemente a livello degli organi di entrata - uscita.

3. I MICROELABORATORI

3-1. Nozioni sulla configurazione

Quando si considera un elaboratore, non è sempre sufficiente conoscere il tipo di macchina utilizzata, ma è ugualmente importantissimo conoscere la configurazione disponibile, cioè l'insieme tra grandezza della memoria, organi periferici e possibili espansioni.

Questa configurazione può essere espressa da uno schema del tipo:

Esempio di configurazione

Qui lo schema corrisponde ad una configurazione tipo di microelaboratore.

Per elaboratori più potenti, la configurazione può includere più organi periferici, sia come numero, che come qualità.

3-2. Configurazione dei microelaboratori

Un microelaboratore è un elaboratore in cui l'unità centrale è un microprocessore, cioè un circuito ad alta integrazione (LSI) che contiene tutte le funzioni di una unità centrale.

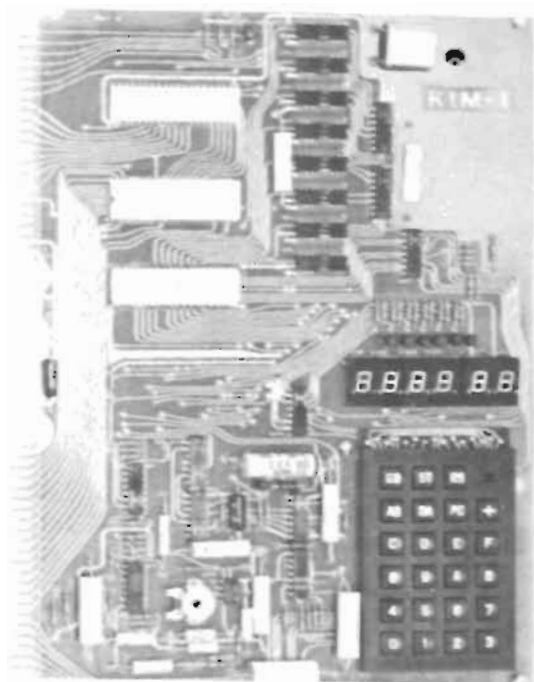
Ci sono differenti categorie di microelaboratori:

Microelaboratori con un solo circuito

Sono dei microprocessori con un programma fissato in memoria morta (ROM) e una piccola memoria viva (RAM). Tali circuiti sono destinati a semplicissime applicazioni sviluppate in grande quantità.

Non sono programmabili in linguaggi evoluti.

Microelaboratori con una sola scheda

Sono formati da schede con circuiti microprocessori, da circuiti di memoria, morta (ROM) e viva (RAM), destinati a contenere programmi e dati e da circuiti di interfaccia di entrata - uscita (PIO).

Una scheda microelaboratore «KIM»

Fino a qualche tempo fa, questi microelaboratori erano programmabili solo in linguaggio macchina, ma attualmente è possibile sviluppare programmi in BASIC (come nell'elaboratore SYM simile a quello presentato in figura, il quale permette, attraverso la connessione ad una tastiera di programmare in BASIC). Esistono comunque dei microelaboratori capaci di essere programmati in linguaggio evoluto PASCAL (scheda WD 9000).

Microelaboratori personali

Sono sistemi compatti e completi, basati su una scheda ma incorporanti una tastiera e una unità di visualizzazione con possibilità di connessione ad organi periferici standard come unità di nastri flessibili, piccole stampanti, cassette magnetiche ... Questi elaboratori possono essere programmati in linguaggio evoluto e particolarmente in BASIC.

Foto di un sistema microelaboratore «PET»

Microelaboratori professionali

Tali sistemi sono, da un punto di vista interno, basati su schede dello stesso tipo che nei microelaboratori personali, ma sono più completi e possono essere connessi a più tipi di unità periferiche professionali (dischi multipli, stampanti rapide, bande magnetiche, connessioni in rete, terminali multipli...).

Di fatto questa categoria di sistemi può essere attualmente confusa con i sistemi di microelaboratori. Essi dispongono di logiche sofisticate: multiprogrammazione, sistemi di gestione, di schedari e possono essere programmati in diversi linguaggi evoluti: BASIC, FORTRAN, COBOL, PASCAL...

Possono essere utilizzati in applicazioni commerciali o industriali, in sostituzione dei sistemi mini elaboratori a causa dei costi di acquisto ben inferiori.

Foto di un sistema 'APPLE'

Tutti i sistemi (ad esclusione dei microelaboratori ad un solo circuito) hanno in comune il fatto che possono attualmente lavorare con il BASIC, cosa che mostra quindi l'importanza del linguaggio nel futuro dei microelaboratori.

Nota. — Sulla questione del sapere dove si ferma la classificazione dei microelaboratori, abbiamo visto come l'ultima categoria ha le stesse funzioni dei minielaboratori. Si può dunque dire che questa frontiera non ha più senso, nella misura in cui nel 1979 esistono sul mercato microprocessori 16 bits potenti come un medio minielaboratore. È certo che attualmente, e ancora più nel futuro, la programmazione di questi microelaboratori sarà effettuata con linguaggi evoluti. I sistemi logici sono sempre più sofisticati e gli utenti più numerosi: ciò fa prevedere che gli stessi linguaggi verranno usati per lo sviluppo di nuove applicazioni. Ciò porterà ad una disponibilità di prodotti logici di massa utilizzabili su una grande quantità di sistemi con costi relativamente bassi in funzione del grande numero di esemplari distribuiti.

Questo è uno dei caratteri fondamentali dell'arrivo sul mercato dei microelaboratori: il «sistema logico» può essere utilizzato su grande scala. Ciò non esclude d'altronde la possibilità di una nuova categoria di «informatici artigiani» e fabbricanti di prodotti logici da vendere ad utenti con particolari necessità.

4. LA NOZIONE DI ALGORITMO

Il compito di un programmatore è essenzialmente di esprimere i problemi da risolvere sotto forma di algoritmo e di tradurli in un certo linguaggio sotto forma di programma.

4-1. Storia ed etimologia

Malgrado le apparenze (algoritmo di Euclide), il termine algoritmo non deriva da una parola greca o latina, ma è una contrazione e una deformazione del nome del matematico arabo Al Khwarismi, che pubblicò due libri importanti: il primo di aritmetica, e il secondo, il cui titolo *Kitab al-jabr wal muqabala* ('metodo per numerare ed ordinare le parti di un tutto') è all'origine della parola algebra.

Quando il primo libro fu tradotto, tre secoli più tardi, in latino, il titolo che gli venne dato fu *Algorismus.* Infatti, a quell'epoca, il termine designava il metodo di numerazione utilizzato ora in aritmetica, e che era stato di fatto scoperto dagli Indi e trasmesso in Europa dagli Arabi. Così, a causa della sua origine, la parola algoritmo ha qualche cosa a che vedere con il significato più conosciuto di procedimento di calcolo.

4-2. Definizione

La prima definizione del termine algoritmo corrispondente al suo senso attuale è data dal matematico Markov:
«L'insieme delle regole precise che definiscono un procedimento di calcolo destinato ad ottenere un determinato risultato, partendo da certi dati iniziali».

Gli algoritmi obbediscono alle tre seguenti proprietà che devono obbligatoriamente possedere:

a) sono costituiti da un insieme di regole precise e sono comprensibili da tutti;
b) si applicano a dati che possono variare in larga misura;
c) tendono a ricavare un risultato, che si ottiene quando i dati son ben scelti.

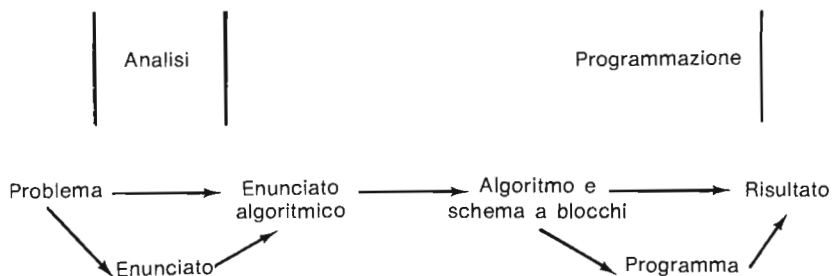
Questa nozione di algoritmo può anche essere assimilata alle nozioni più comuni di «metodo», «tecnica», «istruzioni», «procedimento», ecc... Nel settore dell'informatica, questa definizione è leggermente insufficiente, per cui bisogna aggiungere certi caratteri operazionali quali:

- un algoritmo deve essere finito: è un insieme finito di istruzioni, ciascuna delle quali deve terminare in un tempo finito;
- il procedimento in un algoritmo è di natura discreta e non continua;
- il funzionamento dell'algoritmo è deterministico: deve cioè sempre dare gli stessi risultati per dati che siano gli stessi;

— c'è la presenza di un operatore che effettua le operazioni (l'operatore può essere una persona, una apparecchiatura meccanica o elettrica, ecc...);
— esiste un supporto di memoria che permette di immagazzinare non solo i risultati intermedi e finali, ma anche le regole e i dati.

Attualmente, nel campo dell'informatica, la miglior definizione di algoritmo è stata data da Knuth nella sua opera *L'arte di programmare*:
«È un insieme di regole (o istruzioni) avente le cinque seguenti caratteristiche:
— deve essere finito e terminare dopo un numero finito di operazioni;
— deve essere definito e preciso: ogni indicazione deve essere definita senza ambiguità;
— il campo di applicazione dei dati di entrata deve essere precisato (esempio: numeri interi, reali, negativi, ecc...);
— deve possedere almeno un risultato (dato di uscita);
— deve essere effettuabile: tutte le operazioni devono poter essere effettuate esattamente e in un tempo finito da un uomo che utilizzi mezzi manuali.

4-3. Problemi, algoritmi e programmazione

Assegnato un problema concreto, si inizierà dalla definizione di un enunciato preciso di quel problema (per esempio, calcolare il massimo comun divisore tra due numeri), successivamente, si passerà all'enunciato algoritmico che indicherà le differenti tappe dell'algoritmo, sotto forma di schema a blocchi o di schema di programma che sarà successivamente utilizzato per arrivare al risultato del problema posto.

È necessario tener presente che un dato problema non conduce necessariamente ad un unico algoritmo. Si dirà che un problema è risolubile se esiste almeno un algoritmo che ne permetta la risoluzione. È importante sapere che non tutti i problemi sono risolubili, per questi problemi cioè non esiste algoritmo che dia la soluzione.

Bisogna insistere sul fatto che l'esistenza di una soluzione per un problema, in un caso particolare o in più casi specifici, non implica che il problema

sia risolubile. Inversamente, il fatto di dire che il problema non sia risolubile, non implica il fatto che non abbia soluzioni, ma solo che non esiste un algoritmo tale da trovare le soluzioni in tutti i casi in cui esse esistono.

Oltre a ciò che abbiamo visto, un algoritmo può essere come una serie di operazioni da effettuare per risolvere un problema. Così, un modo di impiego può essere considerato come la descrizione dell'algoritmo di utilizzazione di un apparecchio.

D'altra parte, bisogna ricordare che la maniera di apprendere il calcolo numerico o la stessa analisi grammaticale e l'ortografia nella scuola elementare fà rilevare spesso metodi algoritmici. Speriamo così di aver rassicurato il lettore, in modo che il concetto di algoritmo non lo spaventi più. Nel seguito, verrà dato un certo numero di semplici esempi: essi vanno dal più semplice ad uno di una certa complessità e si vedrà che questi differenti tipi di algoritmi si ritrovano in programmazione.

Bisogna anche avvertire il lettore che *non esistono metodi* per scoprire un algoritmo riguardante un problema non ancora risolto, sia in maniera personale che collettiva. La scoperta di un algoritmo è dunque un atto essenzialmente *creativo*, che dipende non solo dall'intelligenza, ma dall'intuizione e da una certa dose di esperienza.

D'altra parte, uno stesso problema può essere risolto con più tipi di algoritmi. Così, per esempio, si può risolvere in maniera diversa il problema della scelta di numeri in ordine crescente o decrescente. Esistono infatti numerosi algoritmi che danno una soluzione a questo problema.

Bisogna quindi distinguere tre tipi di problemi:

- i problemi risolvibili praticamente in maniera manuale o intellettuale in ogni caso. Allora l'analisi dei metodi utilizzati per risolvere il problema deve permettere la formalizzazione di un algoritmo;
- i problemi per cui è già stato trovato un algoritmo. Può essere allora interessante ricercarne un altro più semplice o che conduca più velocemente al risultato. Si tratta di un lavoro di ricerca informatica.
- I problemi che non si sanno risolvere in tutti i casi o del tutto. In questi casi, l'analisi passa da una fase di ricerca che può non dare risultati. Se il problema è dubbio, non ci sono algoritmi. Se il problema è ben precisato, lo si può inizialmente semplificare e, successivamente, si può cercare di generalizzarlo. In questo caso, il lavoro è molto più difficile e quindi il risultato non è certo.

4-4. I differenti livelli di complessità degli algoritmi

1° La formula

Avendo dato il problema: calcolare il salario giornaliero di una persona,

conoscendo il salario orario ed il numero delle ore di lavoro, questo può essere risolto con l'algoritmo seguente:

Sia H il salario orario.
Sia T il numero di ore.
Allora il salario di base giornaliero sarà $B = H \times T$.

L'algoritmo corrispondente è semplicissimo e può essere riassunto nella maniera seguente:

Dati H, T.
$B = H \times T$.
Risultato B.

In questo caso, una sola formula di calcolo riassume l'algoritmo.

2° La sequenza di formule con risultati intermedi:

Complicando leggermente il problema precedente, ci si può domandare:

«Calcolare il salario netto, conoscendo la paga oraria, il numero di ore ed includendo le trattenute previdenziali, sanitarie e fiscali».

Bisogna allora aggiungere le seguenti affermazioni nell'enunciato algoritmico:

Sia P la percentuale di ritenute SS sul salario di base.
Allora la ritenuta sarà $R = B \times P$.
E il salario netto sarà $N = B - R$.

In modo più sintetico:

Dati H, T, P.
$B = H \times T$.
$R = B \times P$.
$N = B - R$.
Risultato N.

Qui l'algoritmo è una sequenza di più formule con risultati intermedi come B e R.

Si noterà come l'ordine in cui vengono fatte le operazioni corrispondenti alle formule è significativo (così B deve essere calcolato prima di R e R prima di N).

3° Algoritmo condizionale

Complicando ancora l'esempio precedente, si può fissare un tetto per la ritenuta, al di sopra del quale questa non viene aumentata.

Allora l'enunciato algoritmico deve essere completato da:

Sia MAX.
B = H × T.
R = B × P.
se R > MAX; allora R = MAX.
N = B − R.
Risultato N.

Si è qui introdotto un test su una condizione conseguente ad un calcolo.

4° Algoritmo iterativo

Gli algoritmi precedenti funzionano solo per una serie di dati. Se vogliamo ottenere ora la paga di cento persone, utilizzando gli algoritmi precedenti, bisognerà operare una iterazione sull'insieme delle persone. Così l'algoritmo diventerà iterativo. Considerando I come un numero della sequenza degli individui, si ha:

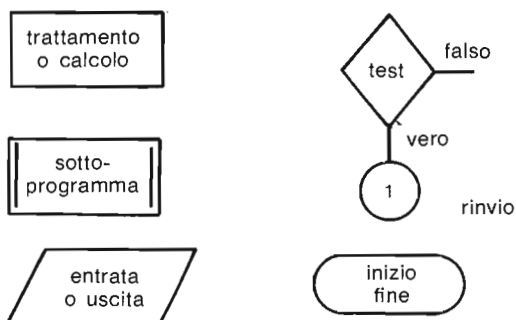
Dati comuni P, MAX.
Per I che varia da 1 a 100.
Dati H (I), P (I).
B (I) = H (I) × T (I).
R (I) = B (I) × P.
Se R (I) > MAX, allora R (I) = MAX.
N (I) = B (I) − R (I).
Risultato N (I).

Con questi esempi, abbiamo visto delle strutture algoritmiche che si possono trovare in tutti i linguaggi di programmazione. Per completezza, bisognerà aggiungere gli algoritmi ricorrenti che utilizzano una definizione che faccia intervenire il risultato dello stadio precedente, per ottenere il risultato attuale.

Così, per esempio, si sa che $n! = (n - 1)! \times n$, e ciò si può scrivere come FATT (N) = N × FATT (N − 1).

5. GLI SCHEMI A BLOCCHI O ORGANIGRAMMI

Esiste un metodo grafico di descrizione degli algoritmi che consiste nel descrivere le differenti operazioni sotto forma di uno schema indicante i differenti ordini o le differenti condizioni che sono trattate dall'algoritmo.

I principali simboli usati sono:

Esempio:

Se, per esempio, si prende l'algoritmo condizionale precedente, avremo:

INIZIO

MAX
H, T, P

Entrano i dati MAX, H, T, P

B = H x T

Calcolo del salario di base

R = B x P

Calcolo della ritenuta

Test per verificare se la
ritenuta supera il tetto

R < MAX

Falso

R = MAX

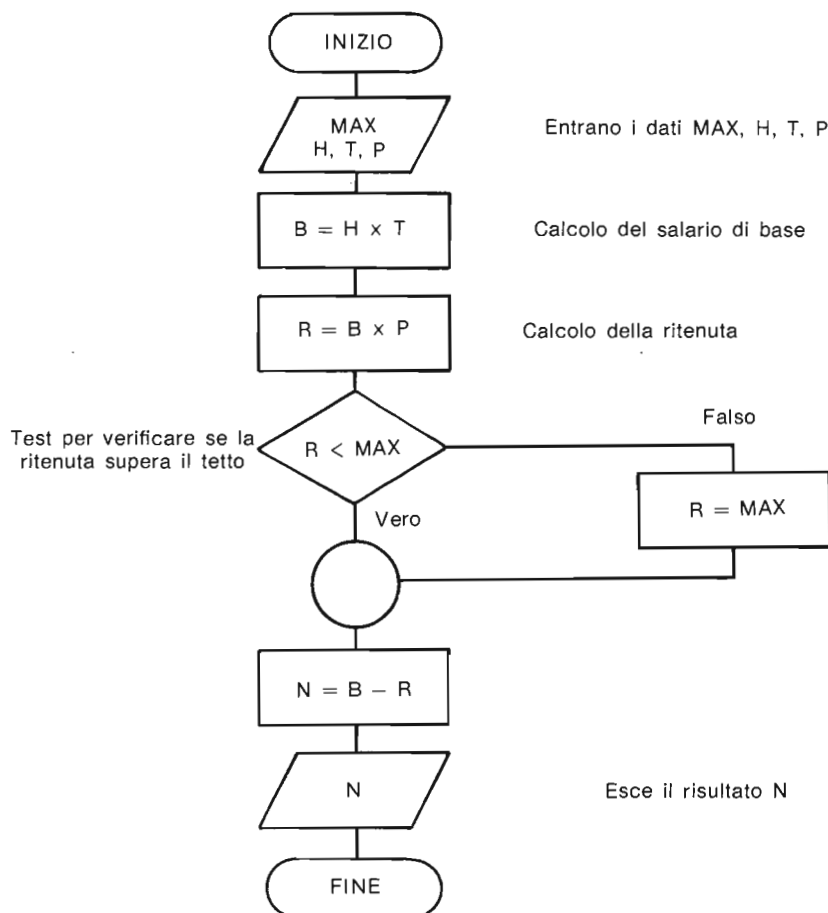
Vero

N = B − R

N

Esce il risultato N

FINE

Utilizzazione degli schemi a blocchi

Inizialmente, in un contesto industriale, la fase di analisi di un problema termina con la stesura di uno schema, che costituisce in qualche modo il progetto di un algoritmo che successivamente potrà essere programmato. Questo schema corrisponde alla necessità di una certa divisione del lavoro in un contesto di produzione.

Lo schema a blocchi serve, in questo caso, come documento di base all'analisi organica, usata come legame tra l'analista e il programmatore. D'altra parte, serve alla documentazione del lavoro dell'analista e teoricamente è indipendente dal tipo di linguaggio di programmazione usato. Per questo motivo lo stesso schema a blocchi può essere programmato in diversi linguaggi.

Tuttavia, in questi ultimi anni, i metodi si sono evoluti e lo schema a blocchi non è più necessariamente il punto di passaggio obbligatorio di un progetto informatico. In effetti, se il problema è complesso, lo schema a blocchi non sta più in una sola pagina e diventa così di difficile lettura. D'altra parte, può rivelarsi errato durante la fase di programmazione, ciò che implica delle messe a punto e dei ritorni a delle fasi precedenti di lavoro.

A causa di tutti questi motivi, uniti all'esistenza di tecniche di programmazione strutturata, lo schema a blocchi non ha più l'importanza che aveva ancora qualche anno fa.

Bisogna notare, tuttavia, che il BASIC non è estremamente strutturato, per cui lo schema a blocchi riveste ancora una certa utilità per la programmazione.

6. IL SOFTWARE E LA PROGRAMMAZIONE

Nei paragrafi precedenti si è visto che le macchine chiamate elaboratori non possono eseguire che un certo numero limitato di operazioni elementari, dette istruzioni o ordini.

La potenza degli elaboratori sta nel fatto che essi possono eseguire lunghe sequenze di istruzioni elementari in un tempo estremamente breve (così, se il ciclo di base dell'unità centrale è dell'ordine dei microsecondi, si potrà eseguire fino a 1 milione di istruzioni al secondo). È chiaro che se si fosse dovuto specificare 1 milione di istruzioni differenti per tenere occupato l'elaboratore per un secondo, il vantaggio non sarebbe senza dubbio stato compensato dal tempo passato per effettuare questo fastidioso lavoro.

La programmazione consiste precisamente nel trovare metodi iterativi che permettano di risolvere alcuni ben determinati problemi, specificando soltanto un numero relativamente limitato di istruzioni, che saranno ripetute molte centinaia o addirittura milioni di volte dalla macchina.

6-1. Necessità di un codice e di un linguaggio simbolico

Il solo «linguaggio» conosciuto dalla macchina è il linguaggio binario: è quindi indispensabile specificare le sequenze di istruzione e i dati sotto una forma più manipolabile dagli uomini.

Si farà ricorso quindi a notazioni simboliche, che permetteranno di descrivere i passaggi logici da effettuare in un linguaggio simbolico.

Esistono più livelli di simbolismo a seconda che si utilizzi un linguaggio specifico o meno.

6-2. Nozioni di istruzioni simboliche

Per programmare il problema di calcolo già visto in esempio, si potrà immaginare un linguaggio simbolico del tipo:

ISTRUZIONE 1: MOLTIPLICARE IL VALORE DELLA MEMORIA B CON QUELLO DELLA MEMORIA P E INSERIRE IL RISULTATO NELLA MEMORIA R DOPODICHÈ ESEGUIRE L'ISTRUZIONE 2.

ISTRUZIONE 2: SE IL RISULTATO R È INFERIORE A MAX ALLORA ESEGUIRE L'ISTRUZIONE 4, ALTRIMENTI ESEGUIRE L'ISTRUZIONE 3.

ISTRUZIONE 3: INSERIRE IL VALORE MAX IN R.

ISTRUZIONE 4: SCRIVERE LA FRASE: "LA TRATTENUTA S.S. È = "; SCRIVERE IL VALORE DI R.

Un simile linguaggio presenta il vantaggio di essere comprensibile a tutti, ma presenta diversi inconvenienti:

- per ogni istruzione elementare è necessaria una frase relativamente lunga;
- bisogna poter decodificare e tradurre ogni singola frase in modo non ambiguo per la macchina (così ogni errore di ortografia o di omissione di un parametro può rendere l'istruzione incomprensibile per la macchina);
- per un problema complesso, la serie di istruzioni sarà relativamente lunga, e sarà difficile seguire il concatenamento logico delle operazioni.

La prima semplificazione possibile è, da un lato, l'uso di una organizzazione sequenziale di istruzioni: così, nell'esempio citato, si sarebbe potuto evitare di indicare nell'istruzione 1 ciò che era necessario in seguito eseguire nell'istruzione 2. Questa organizzazione sequenziale implicita è utilizzata in *tutti* i linguaggi di programmazione: in altre parole, quando si scrive o si legge un programma dall'alto in basso, le istruzioni sono eseguite in sequenza, salvo indicazione contraria (rottura di sequenza).

Così, ora si vede che ogni istruzione simbolica è composta da tre parame-

tri: parametro di identificazione, parametro indicante la o le operazioni o le azioni da effettuare e parametri operatori.

In altri termini l'esempio precedente potrebbe essere così programmato:

1 MOLTIPLICARE B E P E INSERIRE IL RISULTATO IN R
2 SE R < MAX ANDARE ALLA 4
3 SE NO R = MAX
4 SCRIVERE "LA RITENUTA S.S. È = "; SCRIVERE R

A partire da questo semplicissimo esempio, va detto che si può procedere in due direzioni per definire un linguaggio di programmazione. La prima consiste nel voler confrontare il linguaggio simbolico delle istruzioni realmente eseguite dalla macchina.

Linguaggio assembler. – Se la macchina possiede una istruzione di moltiplicazione chiamata simbolicamente MOL, e di sottrazione, chiamate SOT, delle istruzioni di richiamo o di accomodamento in memoria (RCM = richiama dalla memoria o MIM = metti in memoria) – di diramazione in caso di risultati negativi (DRM) e di stampa (PRINT); si otterrà il programma in linguaggio macchina:

linea	istruzione	operando	
I	RCM	B	richiamare B
	MOL	P	moltiplicarlo per P
	MIM	R	mettere in memoria R
	SOT	M	confrontarlo con M
	DRM	N	andare all'istruzione N se M > R
	RCM	M	
	MIM	R	porre R = M
N	PRINT	MESSAGGIO	
	PRINT	R	
MESSAGGIO	CAR	"RITENUTA S.S.="	
	STOP		

L'istruzione CAR definisce una serie di caratteri.

Benchè non sia negli scopi di questo testo, l'esempio riportato sopra illustra ciò che si definisce linguaggio simbolico assemblatore (assembler), nel quale bisogna precisare ogni operazione da effettuare. Questa soluzione presenta l'inconveniente di essere propria della macchina utilizzata e di avere lunghi tempi di programmazione.

Linguaggio evoluto. – L'altra soluzione consiste nel definire un linguaggio

indipendente dalla macchina su cui si vuole eseguire il programma. Nel seguito del libro ci si orienterà verso tale soluzione, cioè verso l'utilizzazione di un linguaggio detto universale o evoluto.

Così, nel caso dell'esempio sopra esposto, si potranno descrivere le operazioni da effettuare con l'aiuto delle istruzioni.

R : = B × P
SE R > MAX ALLORA R : = MAX
SCRIVERE "RITENUTA S.S. ="; R

Ciò è abbastanza simile a un linguaggio universale come il BASIC o il PASCAL. In questo modo si vedono i vantaggi di un linguaggio evoluto o universale, rispetto ad un assembler: concisione, possibilità di utilizzare espressioni simili alla formulazione matematica, comprensione immediata del programma.

Tuttavia, ciò si traduce nella necessità di obbedire a precise regole di sintassi che permettano la traduzione automatica del programma nel linguaggio della macchina. D'altra parte l'esecuzione del programma sarà generalmente più lenta che nel caso si fosse utilizzato l'assembler.

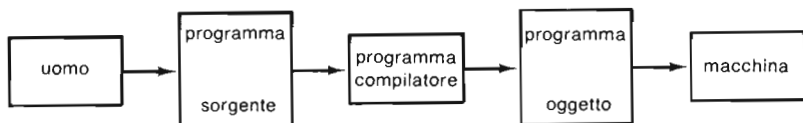
6-3. Nozione di programma sorgente e oggetto: Compilazione o interpretazione.

L'utilizzazione di un linguaggio simbolico, qualunque esso sia (assembler o evoluto), necessita che venga tradotto sotto forma di una sequenza di istruzioni-macchina. Esistono due tipi di traduzione: la traduzione globale, o *compilazione*, e la traduzione istantanea al momento dell'esecuzione o *interpretazione.*

a) *La compilazione*

Il linguaggio in cui è scritto il programma costituisce il *programma sorgente* e la forma in cui sarà eseguito dalla macchina, si chiama *programma oggetto.*

L'operazione di traduzione si chiama compilazione ed è operata da un programma chiamato compilatore.

Schematicamente:

per il programma compilatore, il programma sorgente costituisce i dati e il programma oggetto i risultati. È il programma oggetto ad essere eseguito dalla macchina.

Il programma compilatore è anch'esso eseguito sulla macchina ed in generale è scritto in un linguaggio assembler consono alla macchina..

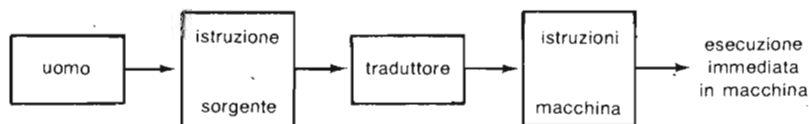
Durante la compilazione, il programma compilatore può trovare e segnalare un certo numero di errori di sintassi o di semantica, che gli impediscono di elaborare senza ambiguità il programma oggetto. In questo caso, bisognerà correggere gli errori, poichè un programma oggetto potrà essere eseguito soltanto quando tutti gli errori saranno stati eliminati.

È importante sottolineare intanto che un programma compilato senza errori, non è obbligatoriamente esente da errori. Infatti, il programma può contenere errori di logica che non possono essere scoperti durante la compilazione.

Per terminare il paragrafo, si dirà che per un linguaggio universale, il programma sorgente può essere tradotto ed eseguito su qualsiasi macchina che abbia un compilatore corrispondente al linguaggio sorgente (cosa che presuppone una standardizzazione dei programmi). Al contrario, l'esecuzione di un programma oggetto non sarà possibile se le macchine non avranno le stesse caratteristiche e la stessa configurazione.

Nella pratica, non è raro che diversi compilatori con lo stesso linguaggio differiscano leggermente da una versione ad un'altra, cosa che si traduce nella mancanza di trasferibilità dei programmi sorgente. Si risolve la questione, accettando di non utilizzare gli standards meno flessibili e di modificare quel numero di istruzioni tali da poter passare da una macchina all'altra.

b) *L'interpretazione*

L'altra tecnica è di considerare che il programma sorgente è tradotto, istruzione per istruzione, al momento della sua esecuzione. Questa operazione è effettuata da una macchina che «interpreta», cioè traduce ogni istruzione in linguaggio macchina e lo esegue immediatamente.

Schema di un traduttore

In questo caso, non esiste un programma oggetto e le istruzioni sono tradotte ogni volta che si controlla ciascuna istruzione. Il solo riferimento è allora il programma sorgente.

Un linguaggio interpretato è generalmente *interattivo*. È il caso, ad esempio del BASIC e dell'APL. Ciò permette in particolare di sviluppare o di ve-

rificare i programmi in modo rapidissimo, senza dover passare da fasi di compilazione, nodi, caricamenti, esecuzioni...

Vantaggi e inconvenienti dei linguaggi compilati e interpretati.

I linguaggi compilati presentano il vantaggio di essere meglio strutturati e, in generale, più leggibili. D'altra parte, siccome sono tradotti una sola volta, danno un programma oggetto che non ha bisogno di essere tradotto per ogni esecuzione. Il tempo di esecuzione è dunque più lento che nei programmi interpretati.

I linguaggi interpretati permettono di sviluppare e modificare un programma in modo interattivo e ciò facilita la messa a punto dei programmi. Ciò si paga però al prezzo di una esecuzione più lenta e, in alcuni casi, di programmi mal strutturati, difficili da comprendere o poco modificabili.

Bisogna poi notare che esistono compilatori BASIC che permettono di rendere meno gravi gli inconvenienti di un linguaggio interpretato nel caso di programmi di produzione... È il caso, in particolare, del C-BASIC utilizzato in applicazioni commerciali.

6-4. I Metodi di sfruttamento degli elaboratori

Si è visto che l'utilizzo di linguaggi simbolici necessita di almeno un programma di base: il compilatore o il traduttore. In realtà, l'utilizzo dei linguaggi evoluti necessita di altri programmi detti utilitari e di biblioteche di sottoprogrammi. Si trovano specialmente tutti i programmi che realizzano entrate-uscite, o che realizzano funzioni matematiche come radici quadre, logaritmi, esponenziali ecc...

È dunque necessario avere un insieme di programmi che costituiscano il sistema di sfruttamento, che amministrino l'insieme delle periferiche e la sequenza delle operazioni da effettuare successivamente alla lettura del programma sorgente, alla compilazione, all'interpretazione, all'esecuzione di un listing, all'esecuzione di un programma oggetto fino alla scrittura dei risultati. Tutte queste operazioni sono realizzate sotto controllo di un programma che viene indicato con il termine *monitor* o, anche, «il sistema». In certi casi, si parla ugualmente di programma supervisore, ma questo nome è, in generale, riservato ai sistemi che lavorano in tempi reali (cioè ogni dato che viene dall'esterno è trattato immediatamente o in un tempo estremamente breve).

Esistono, d'altra parte, dei programmi utilitari tali che le *stampanti* permettono di rientrare in un programma e/o di correggere anche i programmi di assistenza per la messa a punto («debugging»). Infine, se si dispone di memorie secondarie, saranno disponibili dei sistemi di gestione dati su dischi (DOS).

Monoprogrammazione e Multiprogrammazione

Questi termini sono utilizzati per indicare il tipo di sistema utilizzato e non per indicare un metodo o un tipo di programmazione.

Così, un sistema lavora in monoprogrammazione se, ad un certo istante, un solo programma viene trattato dalla macchina (un solo programma è presente in memoria centrale). A causa della grande differenza di velocità delle periferiche e delle unità centrali, può capitare che, durante l'esecuzione entrata-uscita, l'unità centrale sia inattiva, cosa che dà un rendimento di utilizzazione troppo basso quando si hanno molte entrate-uscite. Ciò è un po' molesto nel caso di un sistema interattivo avente un tempo di risposta secondo tempi umani.

Un sistema che lavora in multiprogrammazione utilizza i tempi di inattività corrispondenti a entrate-uscite di un programma, per eseguirne un altro o più. Così aumenta considerabilmente il rendimento di utilizzazione dell'unità centrale, ma nel medesimo tempo si aumenta la complessità del sistema. I sistemi microelaboratori di gamma superiore dispongono di tali sistemi logici.

Nella categoria dei sistemi di multiprogrammazione si hanno ugualmente sistemi in divisione di tempo (time sharing) che permettono un dialogo permanente con la macchina attraverso l'intermediazione dei terminali (macchina da scrivere o visori). Per permettere a ciascuno di avanzare simultaneamente nel proprio lavoro, ciascun utilizzatore è alternativamente servito per un certo piccolo tempo, da cui il nome tempo diviso. Ciò è utilizzato sopra ordinatori più potenti.

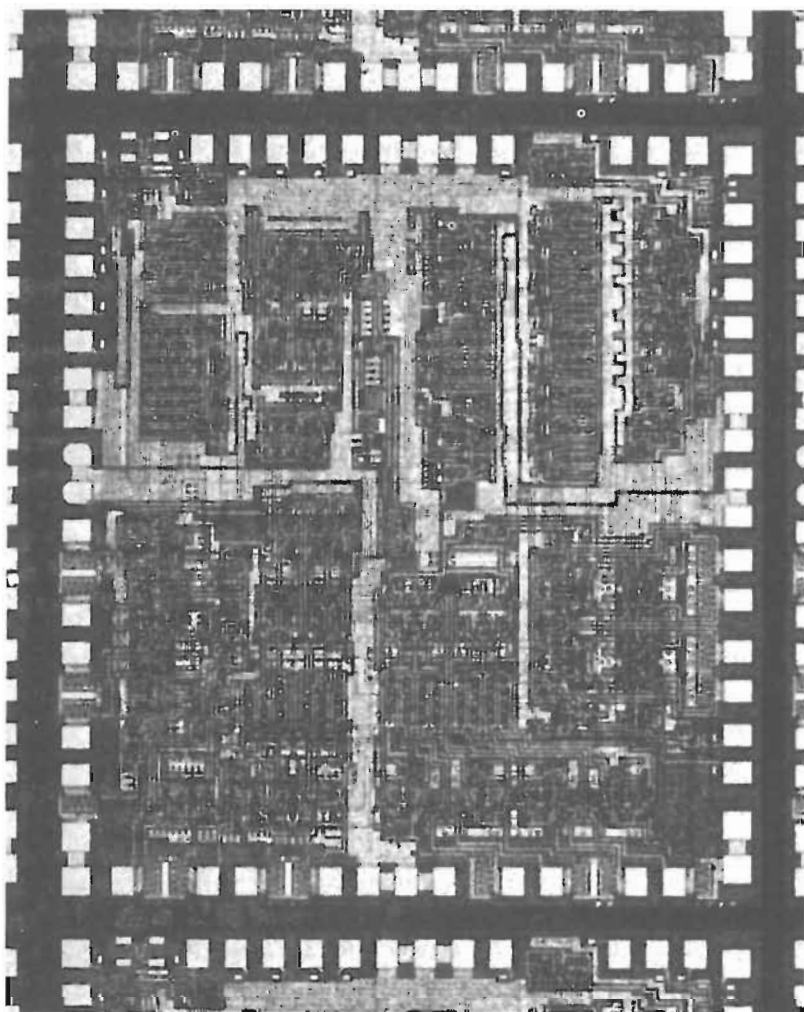
Linguaggio di comando o di controllo

A causa di questi sistemi di sfruttamento più o meno complessi, è anche necessario dare delle istruzioni al sistema, in maniera da indicargli il tipo di lavoro che si vuole effettuare.

Così si hanno dei linguaggi di *comando* che permettono di specificare al sistema che si vuole compilare un programma, successivamente di ottenere un listing ed infine di eseguire un programma.

Sui grossi sistemi non interattivi ciò si traduce in un certo numero di comandi espressi sotto forma di «schede controllo» che si devono piazzare prima e vicino al programma. Uno dei maggiori inconvenienti di questi linguaggi di comando è che non sono standardizzati e che variano da una macchina all'altra.

D'altra parte, quanto alla loro sintassi, sono spesso inutilmente astrusi e rigidi.

Uno dei grossi vantaggi dei microelaboratori è di utilizzare dei linguaggi di comando semplicissimi e quasi trasparenti per l'utilizzatore. Ciò facilita la messa in opera di programmi su questo tipo di macchine.

Una pastiglia (CHIP) ad alta integrazione

CAPITOLO 2

GENERALITÀ SUL LINGUAGGIO BASIC

In questo capitolo presenteremo, riassumendo, le caratteristiche del linguaggio. Ci riferiamo alla versione del BASIC, definita dal Dartmouth College (1960). Benchè esistano variazioni che seguono la tecnologia e i prodotti delle ditte costruttrici, si tratta comunque dello standard di base del linguaggio.

Inizialmente, il linguaggio è stato sviluppato come un utile strumento interattivo per chi inizia a programmare. Ci sono state delle modificazioni dopo la prima definizione (1960), per assicurare una migliore possibilità di trasposizione del linguaggio su diversi materiali e per dare maggior flessibilità. In questo capitolo daremo le caratteristiche principali del linguaggio.

Un programma è un seguito di frasi scritte in un linguaggio definito da un alfabeto e un insieme di regole di formazione delle frasi in quel linguaggio. Le regole costituiscono la sintassi o la grammatica del linguaggio.

Prima di scrivere un programma, è necessario studiare il problema da risolvere e trovare un *algoritmo* che permetta di definire in modo preciso la risoluzione del programma. Lo stesso problema può essere risolto da parecchi algoritmi.

Abbiamo visto che un algoritmo è un insieme di istruzioni che opera su dei dati, per arrivare in un tempo finito a uno o più risultati che risolvono il problema.

Questa serie di istruzioni possono essere svolte dall'ingegno umano, indipendentemente dalla loro scrittura, in un linguaggio di programmazione o della loro esecuzione da parte di una macchina.

1. IL BASIC È UN LINGUAGGIO UNIVERSALE

Il linguaggio BASIC deriva dal linguaggio di programmazione FORTRAN, e può essere considerato, in questo senso, un linguaggio scientifico. Bisogna notare tuttavia che il BASIC dispone di istruzioni di manipolazione

di caratteri molto più pratiche del FORTRAN, cosa che lo rente più utile per le applicazioni di gestione.

Si può dunque dire che il campo di applicazione del linguaggio è completo: problemi scientifici, problemi di gestione, problemi di trattamento di testi, educazione ecc.

D'altra parte, contrariamente a molti linguaggi di programmazione, la messa in opera di piccoli calcoli è semplice e diretta: così il BASIC può essere utilizzato per effettuare operazioni di calcolo elementare in maniera più semplice che su una calcolatrice.

Infine, le istruzioni di entrata e di uscita dei risultati sono ugualmente semplicissime e più flessibili che nella maggior parte degli altri linguaggi. Un programma è composto da una serie di istruzioni che possono essere chiamate le «frasi» del linguaggio.

Studieremo le regole di formazione di queste istruzioni. Daremo poi esempi di ciascun tipo di istruzioni.

2. IL BASIC È UN LINGUAGGIO INTERATTIVO E CONVERSAZIONALE

Il linguaggio BASIC è il primo linguaggio di programmazione interattivo sviluppato già venti anni fà, in un periodo in cui gli elaboratori non erano accessibili, se non attraverso un trattamento per parti («batch processing»).

In questo senso, ha costituito un progresso in rapporto ai linguaggi compilati, poichè ha permesso lo sviluppo e l'utilizzazione dei programmi in maniera conversazionale, a partire da un terminale con stampante e tastiera. In seguito, il BASIC è diventato il linguaggio dei piccoli sistemi conversazionali su minielaboratori o sui più grandi sistemi in divisione di tempo («*time sharing*»).

Il BASIC è poi il primo linguaggio che utilizza un interprete. Esistono molti derivati del BASIC, come il FOCAL o il MUMPS, sviluppato dal costruttore DIGITAL.

Il linguaggio è basato sul concetto di istruzione corrispondente ad una riga battuta su una macchina da scrivere. La fine della riga è rappresentata dal carattere Ritorna all'inizio della linea. Il linguaggio si libera quindi dalla concezione dei linguaggi basati sulla scheda o nastro perforato, tipo FORTRAN o COBOL, che hanno delle zone ben delimitate e rigide.

Nel seguito del libro, il carattere «Ritorna all'inizio della linea» verrà simboleggiato con ®.

Mentre la macchina sta rispondendo, noi utilizzeremo il carattere □, per indicare che il testo successivo è inviato dalla macchina.

Il BASIC è poi un linguaggio che permette di utilizzare un elaboratore come calcolatrice.

3. IL BASIC COME LINGUAGGIO PER CALCOLI ARITMETICI

Supponiamo di essere sotto controllo di un interprete BASIC e di stampare la sequenza:
STAMPA 7 + 5 ® o PRINT 7 + 5 ®
la macchina risponderà attraverso uno schermo o una macchina da scrivere il risultato:
□ 12
Allo stesso modo, se si batte:
STAMPA 7 – 5 ® o PRINT 7 – 5 ®
il risultato, stampato o visualizzato, sarà:
□ 2
Le altre operazioni possibili, naturalmente, sono la moltiplicazione e la divisione:
STAMPA 3 * 6 ® o PRINT 3 * 6 ®
□ 18
STAMPA 10 / 3 ® o PRINT 10 / 3 ®
□ 3.333333
Si vede in questo ultimo esempio che il BASIC, nella versione estesa, permette di eseguire calcoli immediati su cifre decimali o frazionarie. Per indicare la separazione tra la parte intera e quella decimale, la virgola viene sostituita da un punto.
È ugualmente possibile, con il BASIC esteso, domandare il risultato di più operazioni:

Esempio:

```
STAMPA      2 + 7      7 / 2,      5 * 3 ®
  □           9         3.5          15
```

In questo caso, i risultati sono divisi da spazi bianchi e sono stampati nell'ordine in cui le operazioni sono state specificate.
Negli esempi sopraesposti, il linguaggio è stato utilizzato per effettuare calcoli aritmetici semplici, ma è ugualmente possibile ottenere risultati da funzioni matematiche usuali.
Così, se si batte:
STAMPA ABS (– 12) ® PRINT ABS (– 12) ®
si ottiene:
□ 12 cioè il valore assoluto di – 12
Allo stesso modo:
STAMPA RAD (2) ® PRINT SQR (2) ®
darà la radice quadrata del numero tra parentesi:
□ 1.4141...

o ancora:
STAMPA LOG (2) ®　　　　PRINT LOG (2) ®
□ 0.30103

Le altre funzioni matematiche sono ugualmente disponibili (vedi qui sotto).

Sino ad ora, il linguaggio non offre particolari vantaggi rispetto ad una calcolatrice tascabile. Bisogna ora dire che è possibile far stampare una parola, cosa non permessa ad una calcolatrice. In effetti, quando per mezzo della stampante o dello schermo leggiamo un risultato senza nessuna altra indicazione, non possiamo sapere a cosa corrisponda quel risultato.

Innanzitutto possiamo far stampare delle stringhe di caratteri.

Una stringa di caratteri è una serie di caratteri battuti su tastiera e delimitata da caratteri speciali indicanti l'inizio e la fine della serie stessa. I caratteri utilizzati sono le virgolette: " .

Ad esempio «BUONGIORNO» è una stringa di caratteri.
Si può domandare alla macchina di stamparla, battendo:
STAMPA "BUONGIORNO" ®　　　　PRINT "BUONGIORNO" ®
□ BUONGIORNO

Ma, cosa più interessante, si possono combinare assieme testo e calcoli.
Così si può scrivere l'istruzione:
STAMPA "IL CALCOLO DIPENDE DA", 40 * 0.176 ®
□ IL CALCOLO DIPENDE DA 7.04

Per ora ci fermiamo a questi esempi di uso del BASIC come linguaggio utile al calcolo diretto. In effetti, la potenza del linguaggio deriva dalla possibilità di scrivere una serie di istruzioni costituenti un programma.

3-1. La nozione di istruzione in BASIC

Nei paragrafi precedenti si sono visti degli esempi di istruzione diretta che costituisce una frase del linguaggio: questo testo comprende delle parole specifiche di linguaggio (parole chiave o parole riservate), dei segni di punteggiatura, dei nomi specificati dal programmatore, delle cifre, degli operatori, delle stringhe di caratteri.

Questo insieme obbedisce ad un certo numero di regole di composizione e di *sintassi.*

Apprendere un linguaggio di programmazione è dunque, prima di tutto, apprendere la sua sintassi, ma ciò non è certamente sufficiente per risolvere un problema!

Nel BASIC ogni istruzione comprende un *numero di identificazione*, seguito dal testo dell'istruzione. Nello stesso programma, due istruzioni non possono avere lo stesso numero. Questi numeri di identificazione si chiamano anche etichette.

Esempio

```
10      STAMPA "LA SOMMA DI 4 + 5 VALE", 4 + 5
(10     PRINT "LA SOMMA DI 4 + 5 VALE", 4 + 5)
```

10 è l'etichetta.

- STAMPA (PRINT) è una parola chiave del linguaggio.
- "LA SOMMA DI 4 + 5 VALE" è una sequenza di caratteri da stampare.
- 4 + 5 è una sequenza di cifre separate da un operatore indicante che si deve eseguire una somma.

In questo caso, si tratta di una istruzione semplicissima, destinata a stampare un messaggio associato ad un risultato di calcolo aritmetico (qui si tratta dell'addizione di 4 + 5).

Il risultato di questa operazione si tradurrà in un calcolo che porta a:

```
□   LA SOMMA DI 4 + 5 VALE 9
```

L'interesse di tale istruzione è certamente limitato, nella misura in cui bisognerà scriverla di nuovo ogni volta che si vorrà eseguire una banale addizione!

3-2. La nozione di variabile

Nello stesso modo in cui si apprende a contare con delle cifre e in cui si passa successivamente a simboli per rappresentare variabili suscettibili di assumere diversi valori, in programmazione si utilizzano delle variabili per rappresentare valori (o dati) che cambieranno ad ogni esecuzione di programma.

Così possiamo considerare due variabili X e Y, per rappresentare due numeri interi o reali e per definire la somma, attraverso la relazione:

$$S = X + Y$$

In BASIC si può scrivere:

```
10      SIA S = X + Y
20      STAMPA "LA SOMMA DI X + Y VALE", S
```

o, in inglese:

```
10      LET S = X + Y
20      PRINT "LA SOMMA DI X + Y VALE", S
```

Le due istruzioni non bastano per ottenere un risultato, in quanto X e Y non sono dei valori definiti.

3-3. La nozione di dato

Se si vuole effettuare, come abbiamo fatto precedentemente, una operazione su numeri, bisognerà dare dei valori a X e Y.

Possiamo allora aggiungere le seguenti istruzioni:

```
 1      SIA X = 4
 2      SIA Y = 5
10      SIA S = X + Y
20      STAMPA "LA SOMMA DI X + Y VALE", S
30      FINE
```

o, in inglese:

```
 1      LET X = 4
 2      LET Y = 5
10      LET S = X + Y
20      PRINT "LA SOMMA DI X + Y VALE", S
30      END
```

Questa è una serie di istruzioni secondo una certa logica, quindi darà un risultato corretto.

Se poi si desidera effettuare l'operazione 124 + 385, bisognerà scrivere di nuovo le istruzioni 1 e 2. In questo modo, però, il vantaggio di avere introdotto delle variabili è perso, poichè si assegnano a queste variabili dei valori costanti.

Per avere più flessibilità, bisognerà considerare X e Y come delle variabili contenenti dei valori che verranno dati alla macchina al momento dell'esecuzione del programma. I dati sono proprio questi.

Un *dato* è quindi un valore che l'utente, e non il programmatore, darà alla macchina al momento dell'*esecuzione* del programma.

Così, nel BASIC, si può chiedere l'ingresso dei dati destinati alla elaborazione da parte del programma.

Le istruzioni 1 e 2 sono allora sostituite dalla istruzione ENTRA (INPUT).

```
 1      ENTRA X, Y
10      SIA S = X + Y
20      STAMPA "LA SOMMA DI", X, "+", Y, "VALE", S
30      FINE
```

o, in inglese:

```
 1      INPUT X, Y
10      LET S = X + Y
20      PRINT "LA SOMMA DI", X, "+", Y, "VALE", S
30      END
```

Al momento dell'esecuzione del programma, la macchina chiederà l'ingresso dei dati X e Y, mediante l'uso di un punto interrogativo: ? .

Se si risponderà 4 seguito da un ritorno del carrello (tasto RETURN), questo corrisponderà al valore X, la macchina darà allora il secondo punto

interrogativo e, se si risponderà 5 seguito dal ritorno del carrello, allora l'esecuzione del programma continuerà, dando un risultato, peraltro scontato:

LA SOMMA DI 4 + 5 VALE 9

Il vantaggio sta nel fatto che, questa volta, se si esegue di nuovo il programma, inserendo i dati 124 e 385, si otterrà:

LA SOMMA DI 124 + 385 VALE 509

senza modificare il programma.

Questa ultima versione costituisce un programma generale che permette di addizionare due numeri qualunque.

3-4. La documentazione dei programmi

Un programma semplice come quello sopra presentato non necessita di spiegazioni supplementari. Al contrario, se i programmi comprendono diverse decine o centinaia o migliaia di istruzioni, diventa difficile per un utente, o un programmatore, comprendere il programma. In questo caso si possono inserire delle istruzioni che non sono eseguite dalla macchina, ma che servono a documentare e/o a spiegare ciò che fa ogni singola parte del programma.

Questa istruzione è preceduta dalla parola chiave REM (che significa remark, cioè nota), seguita da un testo libero, che chiarisce le operazioni successive.

Esempio:

```
10        REM INPUT DEI DATI
20        INPUT X, Y
30        REM CALCOLO DELLA SOMMA
40        SIA S = X + Y
50        REM SCRITTURA DEI RISULTATI
60        PRINT "LA SOMMA DI X + Y VALE", S
70        END
```

3-5. Utilizzazione di un interprete BASIC

Presenteremo qui in maniera succinta i principali comandi di cui parleremo dettagliatamente alla fine del capitolo.

Il BASIC è un linguaggio interpretato. Benchè esistano in molti casi dei compilatori BASIC, supporremo che si tratti di un sistema interprete. Nella maggior parte dei microelaboratori, l'interprete è immagazzinato in una me-

moria morta (ROM) e, dal momento della messa sotto tensione, il controllo è dato dall'*editore*. Il sistema risponde allora con un PRONTO (READY), o con qualcosa di simile.

L'editore è un programma che permette di inserire alcune linee di testo, e, in particolare, linee di istruzioni BASIC corrispondenti ad un programma.

Gli editori disponibili sono più o meno sofisticati. In generale, essi permettono di inserire delle linee di testo che sono convalidate quando si batte il tasto ® (RETURN). In certi casi l'editore numera le righe, ma, più soventemente, i numeri devono essere introdotti dall'utente: ciò rende il programma più flessibile.

a) *Caso di un nuovo programma*

Se si desidera inserire un nuovo programma, si premerà allora un comando speciale NUOVO (NEW)®, subito dopo il messaggio PRONTO (READY). Se si preme direttamente una istruzione sulla tastiera, questa istruzione sarà presa in considerazione dal momento in cui verrà premuto il tasto®. Così, se la macchina sta per essere fatta funzionare, non è necessario premere il comando.

b) *Caso di un programma già in memoria*

In questo caso, ogni nuova istruzione battuta sulla tastiera, sarà inserita nel programma esistente. In particolare, se l'istruzione è etichettata con un numero già adoperato, questa prenderà il posto della vecchia istruzione avente quel numero.

c) *Elenco di un programma*

Un programma può essere creato e modificato in più momenti. Se si desidera ottenere l'ultima versione, basta utilizzare il comando LIST, che permette di ottenere l'elenco, la lista, delle istruzioni, classificate secondo l'ordine crescente dell'etichettatura.

Ciò permette di verificare la completezza e la correttezza del programma, prima di utilizzarlo.

d) *Partenza di un programma*

Appena si giudica corretto un programma, questo può essere eseguito, utilizzando un comando che chiede l'esecuzione e l'interpretazione del programma. Questo comando è l'ESEGUI (RUN), premuto il quale, l'interprete comincia a tradurre le istruzioni e ad eseguirle. Solo ora l'utente sa che il programma è corretto dal punto di vista sintattico e logico.

Esempio:

Se si vuole scrivere la sequenza di operazioni:

$$15 + 7 \text{ e } 15 - 7$$

si scriverà:

```
10          PRINT 15 + 7
20          PRINT 15 – 7
```

seguite da ESEGUI (RUN)

Si avrà allora l'interpretazione e l'esecuzione dell'istruzione 10, successivamente dell'istruzione 20, e poi avremo il risultato

□ 22
  8 seguito da PRONTO (READY)

e) *Soppressione e inserimento di istruzioni*

Per *sopprimere* una istruzione, basta battere il numero dell'istruzione, seguito da RETURN®.

Per *inserire* una istruzione, basta definirla con un numero tale da inserirsi tra due istruzioni già definite.

Ciò chiarisce il perchè della numerazione di 10 in 10, in quanto essa permette l'ulteriore inserimento di istruzioni.

4. IL BASIC È UN LINGUAGGIO ALGORITMICO

I linguaggi che permettono di trascrivere algoritmi eseguibili da una macchina sono detti algoritmici. Il BASIC è dunque un linguaggio algoritmico.

Esempio:

La nozione di algoritmo non è nuova: si conosce infatti il famoso algoritmo di Euclide per calcolare il MCD (massimo comun divisore) tra due numeri. L'algoritmo può essere espresso con le istruzioni da I_1 a I_5:

I_1 Siano a e b due numeri interi ($a > b$);
I_2 Si divida a per b e sia r il resto ($0 < r < b$);
I_3 Se r vale 0, allora l'algoritmo è terminato e b è il MCD;
I_4 Altrimenti si sostituisce a con b e b con r;
I_5 Continuare, ripartendo da I_2.

Si può allora notare che in questo algoritmo le istruzioni da I_1 a I_5 non sono tutte delle stesso tipo. I_1 è una ipotesi o una affermazione, I_2 è una istruzione di calcolo aritmetico, I_3 è una istruzione condizionale (se... allora..., altrimenti), I_4 è una istruzione di manipolazione di variabili, I_5 è una istruzione senza condizioni.

Questi differenti tipi di istruzione si ritrovano tutti in tutti i linguaggi di programmazione algoritmica.

4-1. Dichiarazioni e istruzioni eseguibili

Un algoritmo è composto da due tipi di istruzioni:
- istruzioni di descrizioni di *dati*, chiamate *dichiarazioni* o definizioni;
- istruzioni di descrizione di *azioni* da effettuare, chiamate *istruzioni* eseguibili.

Il BASIC è un linguaggio con pochissime istruzioni di dichiarazione, in quanto la maggior parte di queste sono implicite. Nell'esempio precedente, si può notare che *a* e *b*, considerati come dati in I_1, diventino variabili in I_4.

Così, in un programma, i dati sono rappresentati dai *valori delle variabili*; queste variabili sono manipolate nelle istruzioni che si eseguiranno.

4-2. Identificatori e parole chiave

Nel linguaggio della programmazione, queste variabili sono rappresentate da *identificatori* che danno loro il nome.

Le regole di formazione di questi identificatori sono semplicissime, per cui si ha una grande libertà di scelta. Nel BASIC, questi sono costituiti da un carattere alfabetico, seguito o meno da caratteri numerici.

Al contrario, in tutti i linguaggi di programmazione, esiste un certo numero di *parole chiave* dal senso ben definito, che possono essere utilizzate solo in una precisa struttura sintattica del linguaggio. Successivamente verrà dato l'elenco esauriente delle parole chiave BASIC.

4-3. Le dichiarazioni in BASIC

Nel BASIC non si hanno istruzioni esplicite di dichiarazione che diano il tipo di dato o di variabile.

Il *tipo di dato* è l'insieme dei valori che possono essere assunti da una variabile. Abbiamo visto che nel BASIC esistono due differenti tipi elementari di dati e variabili: i dati numerici e i dati tipo *stringhe di caratteri.*

Esempio:

Se una variabile assume valori numerici, il suo tipo sarà definito come intero o reale, a seconda del tipo di valori numerici trattati.

In BASIC, i dati sono sia di tipo *scalare*, cioè rappresentati da un insieme di valori elementari, sia di tipo *strutturato*, tali cioè da definire un insieme di valori aggregati sotto forma di elenco o di tabelle. Bisogna tuttavia notare che le tabelle sono, al massimo, di due dimensioni.

a) *Dichiarazione di tipo scalare*

Esistono due tipi di dati: i *numerici* e le *stringhe di caratteri.*

Questi due tipi di dati non devono essere necessariamente dichiarati, ma è l'identificatore della variabile che ne determina il tipo. Così, se l'identificatore è seguito dal carattere **$**, sarà considerato come una catena di caratteri. Allo stesso modo, se la variabile sarà seguita da %, la variabile sarà considerata come numero intero.

b) *Dichiarazione di tipo strutturato*

Esistono due tipi di strutture di dati: le *tabelle di valori numerici* e le *tabelle stringhe di caratteri.*

In una tabella o matrice, tutti i componenti sono dello stesso tipo. Un elemento di matrice è indicato da un indice che deve essere un numero intero. Questo indice permette di poter accedere a qualunque elemento della matrice nello stesso tempo. Nel caso di dichiarazione del tipo strutturato, si dovranno dare le dimensioni della matrice.

4-4. Le istruzioni eseguibili

a) *Istruzioni di calcolo*

Come si è visto nell'esempio dell'algoritmo precedente, le istruzioni sono di due tipi: le istruzioni di calcolo e le istruzioni di transfert o di controllo del programma.

Per un linguaggio di programmazione evoluto, bisogna aggiungere delle istruzioni di entrata-uscita. Così, nel BASIC, come in tutti i linguaggi odierni di programmazione, le istruzioni di calcolo sono istruzioni di *assegnazione.*

Una istruzione di assegnazione è una istruzione che comprende due parti: una parte sinistra e una parte destra, separate da un operatore di assegnazione (=). La parte destra precisa il calcolo da effettuare, mentre la parte sinistra indica la variabile utilizzata per memorizzare il risultato del calcolo precisato nella parte destra.

Esempio:

```
F = A * B + 3
```

Qui, il calcolo effettuato è $a \times b + 3$ ed il risultato è assegnato (memorizzato) nella variabile F.

Torneremo dettagliatamente su queste istruzioni e sulle regole di sintassi ad esse associate nei prossimi capitoli.

Riprendiamo gli esempi di algoritmo dati nel primo capitolo.

I dati H e T rappresentano il numero di ore e la tariffa oraria: questi sono dati numerici. Si può dunque considerarli come variabili numeriche chiamate H e T. Il risultato, ugualmente numerico, è chiamato B.

10	ENTRA H, T	10	INPUT H, T
20	SIA B = H * T	20	LET B = H * T
30	STAMPA B	30	PRINT B
40	FINE	40	END

Se si vogliono calcolare le trattenute S.S. e lo stipendio netto, si aggiungerà semplicemente un dato P e due istruzioni di calcolo che permetteranno di calcolare le trattenute ed il salario.

In questa maniera, tuttavia, il dato P non è modificabile. Può essere considerato quindi come una costante.

```
10    SIA P = 0.04        percentuale del 4%
20    ENTRA H, T
30    SIA B = H * T
40    SIA R = B * P
50    SIA N = B - R
60    STAMPA B, R, N
70    FINE
```

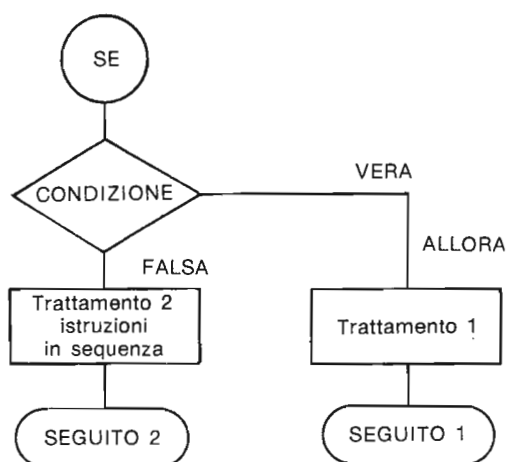
o, in inglese:

```
10    LET P = 0.04
20    INPUT H, T
30    LET B = H * T
40    LET R = B * P
50    LET N = B - R
60    PRINT B, R, N
70    END
```

b) *Le istruzioni di transfert*

Le istruzioni di *transfert o di controllo*, permettono sia di effettuare dei *test* su certe condizioni che indirizzano verso differenti trattamenti del programma, sia di controllare una istruzione particolare del programma.

In questa categoria si hanno le *istruzioni di test o condizionali*, che hanno una struttura comunissima in tutti i linguaggi di programmazione: si trat-

ta del SE... ALLORA (IF... THEN), rappresentato dal seguente schema a blocchi:

L'altra istruzione di transfert è quella che provoca un salto incondizionato ad una istruzione di programma: è l'istruzione VAI A (GO TO).

Esempio di programmazione di un algoritmo condizionale

Riprendiamo l'esempio n. 3 del capitolo 1.

Il dato supplementare è il valore del tetto massimo di trattenute che chiameremo M.

Bisogna dunque verificare se la trattenuta è superiore a questo massimo M. Ciò verrà eseguito con una istruzione di test SE... ALLORA.

Si ottiene quindi il programma:

```
10        SIA P = 0.04
20        SIA M = 200
30        ENTRA H, T
40        SIA B = H * T
50        SIA R = B * P
60        SE R > M ALLORA SIA R = M
70        SIA N = B - R
80        STAMPA B, R, N
90        FINE
```

cioè in BASIC:

```
10        LET P = 0.04
20        LET M = 200
30        INPUT H, T
40        LET B = H * T
50        LET R = B * T
60        IF R > M THEN LET R = M
70        LET N = B – R
80        PRINT B, R, N
90        END
```

Si può ugualmente programmare questo algoritmo, utilizzando l'istruzione VAI A (GO TO).

```
 10       SIA P = 0.04
 20       SIA M = 200
 30       ENTRA H, T
 40       SIA B = H * T
 50       SIA R = B * P
 60       SE R > M ALLORA VAI A 100
 70       SIA N = B – R
 80       STAMPA B, N, R
 90       STOP
100       SIA R = M
110       VAI A 70
120       FINE
```

Questo programma è ugualmente corretto, ma è più confuso e complesso: è stato svolto per mostrare l'uso del salto incondizionato che in certi casi è necessario.

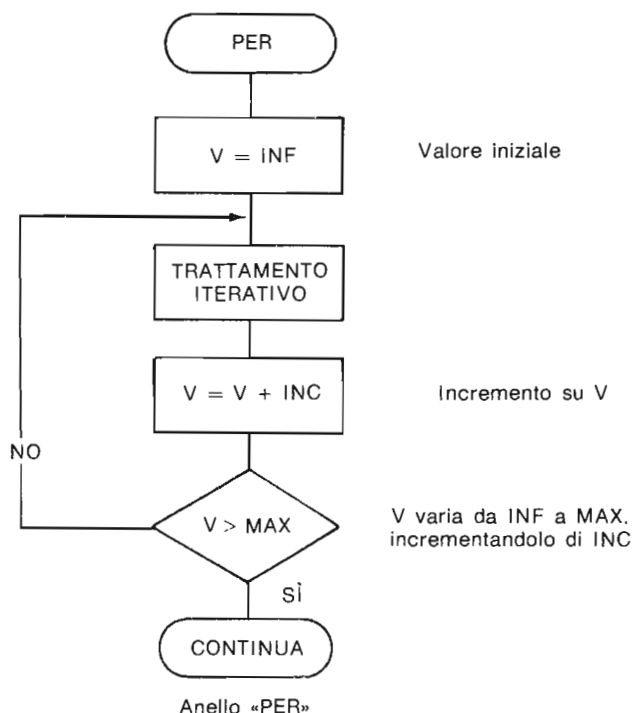
Si è vista poi anche l'istruzione STOP, che permette di fermare l'esecuzione del programma, in modo che non si vada all'istruzione 100, se non prima di avere stampato i risultati richiesti.

c) *Le strutture di algoritmo iterative o ripetitive*

Questa ultima struttura permette di fare un certo numero di iterazioni, utilizzando una variabile di controllo in cui si fissano i limiti delle variazioni. Si tratta dell'anello di programma usatissimo in tutti i linguaggi (anello PER «DO», «FOR»)...

Esempio:

Per V = INF A MAX FARE...

Questa struttura può essere rappresentata dal seguente schema a blocchi:

Anello «PER»

In BASIC l'anello PER può essere effettuato anche utilizzando incrementi negativi. In questo caso, la variabile di controllo decresce da un valore massimo ad un valore minimo.

Esempio di programmazione di algoritmo ripetitivo

In questo caso, i dati e le variabili sono le stesse, ma si domanda alla macchina di effettuare gli stessi calcoli per 10 individui (I).

Si ottiene il programma:

```
10    SIA P = 0.04
20    SIA M = 200
30    PER I = 1 A 10
40    ENTRA H, T
50    SIA B = H * T
60    SIA R = B * P
70    SE R > M ALLORA R = M
```

```
80        SIA N = B − R
90        STAMPA B, R, N
100       PROSSIMO I
110       FINE
```

Si vede qui un esempio di istruzione PER, che permette di far eseguire 10 volte le istruzioni da 40 a 90.

L'istruzione «PROSSIMO I» indica che si passa all'iterazione seguente, cioè che il valore di I è incrementato di 1 per ogni iterazione e che ciò avviene fino a quando I assume il valore 10.

Con notazione inglese si avrà:

```
10        LET P = 0.04
20        LET = 200
30        FOR I = 1 TO 10
40        INPUT H, T
50        LET B = H * T
60        LET R = B * P
70        IF R > M THEN LET R = M
80        LET N = B − R
90        PRINT B, R, N
100       NEXT I
110       END
```

Nota. — Si sono viste, in questo paragrafo, le caratteristiche essenziali del linguaggio, ma ciò rappresenta sicuramente soltanto una visione superficiale sulle possibilità proprie del linguaggio. In effetti, esistono altre istruzioni, in particolare la possibilità di definire sottoprogrammi, di lavorare su stringhe di caratteri, su tabelle, e, in alcuni casi, su matrici. Nei seguenti capitoli vedremo più dettagliatamente queste istruzioni; per ora è utile precisare i comandi necessari per l'utilizzazione di un sistema BASIC.

5. I LINGUAGGI DI COMANDO IN UN SISTEMA BASIC

Abbiamo visto che esistono metodi di sfruttamento per amministrare le differenti risorse di un elaboratore. Si tratta dei sistemi «monitors» o supervisori.

Quando si utilizza un elaboratore che dispone di un interprete BASIC, è necessario poter dialogare con questo sistema di sfruttamento mediante l'aiuto di comandi conosciuti dal monitor.

I comandi non fanno parte del linguaggio e variano da un sistema all'altro.

In questo senso, i comandi che presentiamo, sono il minimo necessario per sviluppare ed eseguire un programma.

Questi comandi corrispondono, da un lato, a parole o frasi a base di parole chiave e vengono prese in considerazione quando si preme il tasto ®come RETURN.

Esistono poi dei comandi di edizione che permettono di fare delle modifiche su un testo già inserito. Questi comandi di edizione corrispondono a caratteri speciali.

5-1. I comandi al sistema

Questi comandi permettono di definire differenti modi di funzionamento del monitor.

Generalmente quando si mette sotto tensione il sistema o dopo aver dato un certo numero di codici di identificazione su di un sistema a tempo diviso, il monitor indicherà che è pronto a ricevere dei comandi scrivendo: PRONTO (READY), o semplicemente con un carattere speciale come], > od altri caratteri a seconda del sistema.

A partire da questo momento, l'utente può battere il tasto di istruzione BASIC o di comando al monitor. La prima scelta è differente dalla seconda in quanto le istruzioni BASIC cominciano sempre con un numero di istruzioni (tranne le istruzioni dirette come PRINT, che possono essere, in una certa misura, considerate come dei comandi al monitor).

a) *L'inizio di un nuovo programma*

Se si vuole cominciare un lavoro al terminale, si deve mettere la macchina sotto tensione; a questo punto la memoria disponibile per l'utente è libera e quindi si possono battere le istruzioni. Se, al contrario, si sta già lavorando e si desidera scrivere un altro programma, si premerà un comando indicante che si incomincia un nuovo programma.

È il comando:

NUOVO (NEW) ®

Questo comando cancella il contenuto di tutto il programma precedente in memoria.

b) *L'elenco di un programma o delle istruzioni*

Se si desidera visualizzare la serie delle istruzioni di programma attualmente in memoria, è sufficiente dare il comando:

LIST ®

che rende noto l'insieme del programma.

Se si desidera conoscere una sola istruzione, si utilizzerà il comando:

LIST *n* ®

dove *n* è il numero della linea.

Così, LIST 140®, darà l'istruzione 140.

Nel caso in cui si desidera elencare un insieme di linee consecutive, si utilizzerà il comando:

LIST *n* 1 – *n* 2 ®

dove *n* 1 è la prima istruzione da elencare e *n* 2 è l'ultima.

In alcuni casi, LIST – *n* 1®, permette di elencare tutte le istruzioni fino alla linea *n* 1 e LIST *n* 1 –®permette di elencare tutte le linee, partendo dalla *n* 1 fino alla fine del programma.

Esempio:

LIST 100 – 300®permetterà di elencare tutte le istruzioni tra 100 e 300.

Nota. – In questo caso, il carattere separatore utilizzato è una virgola o un altro carattere.

c) *Annullamento di istruzioni*

Questa operazione è fatta in funzione del tipo di sistema. Nel caso esista, si tratta del comando:
ANNULLA *n*®
(DELETE *n* o DEL *n*)

In alcuni sistemi basta specificare il numero della linea seguito da ®, e si ha lo stesso effetto.

Esempio:

10 ® cancella la linea 10

Se si dispone di un comando per cancellare, si potrà anche farlo per più linee consecutive:

DEL *n*1, *n*2 ®

cancella tutte le istruzioni da *n*1 a *n*2.

d) *Esecuzione di un programma*

Un programma che si trova già in memoria può partire, premendo il comando ESEGUI:

RUN ®

Nel caso in cui si voglia mettere in moto il programma da una precisa istruzione, si specificherà:

RUN $n1$ ®

che farà partire il programma dall'istruzione $n1$.

Esempio:

RUN 100 ®

fa partire il programma dall'istruzione 100.

e) *Salvaguardia di un programma sulla memoria secondaria*

Questa operazione presuppone che il sistema disponga di una memoria di tipo cassetta magnetica o nastro (rigido o flessibile).

In questo caso, si utilizza il comando SALVA («SAVE»), seguito dal nome del programma o dell'archivio.

Così, per esempio:

SAVE "PROG 1" ®

permette di memorizzare il programma in questo momento in memoria dandogli il nome PROG 1 sul supporto di memoria secondaria utilizzata.

Questo comando può variare a seconda del tipo di macchina utilizzata, tuttavia, la parola chiave SAVE è la più utilizzata.

f) *Caricamento di un programma a partire da una memoria secondaria*

È l'operazione inversa della precedente. Permette di leggere un programma che si trova in uno schedario nella memoria secondaria e di caricarlo in memoria. Tutto ciò viene effettuato per mezzo del comando CARICA (LOAD).

Esempio:

LOAD "PROG 2" ®

permette di caricare il programma PROG 2 nella memoria centrale.

Nota. — Disponendo di più unità di memoria secondaria, bisognerà precisare il numero dell'unità in cui si trova il programma da caricare.

Per la sintassi esatta del programma, bisogna tener presenti i manuali di istruzione delle macchine. Nel caso di programmi in divisione di tempo, bisognerà precisare altri parametri.

g) *La reinizializzazione delle variabili di un programma*

Quando si esegue un programma più volte, tra una esecuzione e l'altra, i contenuti di alcune variabili sono conservate. Se si vogliono rimettere a 0 i valori di tutte le variabili di un programma, si utilizza il comando METTI A ZERO.

CLEAR o CLR ®

Questo comando permette dunque di eseguire di nuovo un programma con le stesse condizioni iniziali.

h) *La continuazione di un programma*

L'ultimo comando importante di un sistema BASIC è quello che permette la continuazione di un programma dopo un arresto (istruzione STOP).

Per ripartire con il programma, si utilizzerà il comando CONT®.

Questo comando avrà dunque come effetto la continuazione del programma dal punto in cui la macchina aveva letto l'ultima istruzione STOP.

i) *L'arresto del programma in corso*

Può essere effettuato con un tasto speciale, chiamato STOP o RESET, oppure con l'aiuto di una combinazione di caratteri di controllo (CONTROL) chiamato *c* e associato generalmente alla lettera C (CONTROL C scritto anche C_c).

5-2. I comandi di edizione

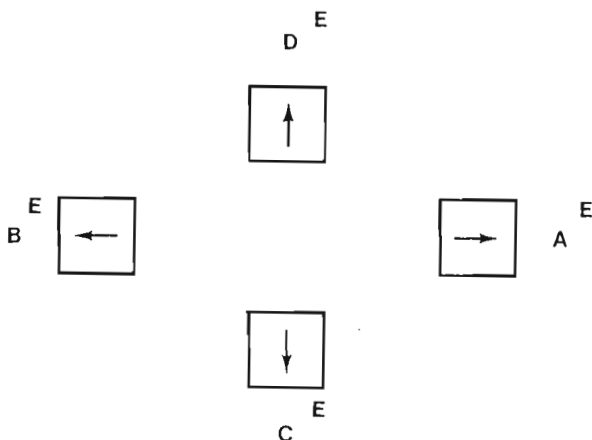
Quando si vogliano modificare le istruzioni di un programma, è necessario poter disporre di comandi che permettano l'edizione delle istruzioni già inserite.

Ora, si è già visto che l'inserzione di una linea può essere fatta con il dare alla linea un nuovo numero, compreso tra i numeri delle linee tra le quali si vuole fare l'inserimento: ciò spiega perchè sia più prudente numerare le istruzioni di 10 in 10, in modo tale da poter inserire facilmente nuove istruzioni.

La cancellazione di una linea è già stata vista. Di contro, cosa molto utile, c'è la possibilità di modificare una istruzione, senza cancellarla totalmente. Per questo scopo esistono due possibilità: o utilizzare i comandi di un editore che non è specifico del sistema BASIC (è il caso dei sistemi a tempo parziale che utilizzano più tipi di linguaggi), o utilizzare l'editore del BASIC.

Nella maggior parte dei sistemi microelaboratori, le funzioni di edizione sono disponibili direttamente in tastiera, grazie alle possibilità di movimento di un cursore su uno schermo di visualizzazione.

a) *Movimento del cursore*

I movimenti del cursore sono possibili grazie a dei tasti specifici che permettono il suo movimento in alto, in basso, a destra e a sinistra. Essi sono in generale rappresentati da tasti con frecce associate ai seguenti movimenti:

Quando questi tasti non sono disponibili, è possibile ottenere lo stesso effetto, utilizzando un tasto di controllo, chiamato ESC (ESCAPE), associato ad una lettera: la E. Il codice ASCII è il 27.

Più soventemente, si hanno le seguenti combinazioni:

- A^E permette il movimento del cursore a destra;
- B^E lo muove a sinistra;
- C^E lo muove in basso;
- D^E lo muove in alto.

b) *L'inserzione e l'annullamento di carattere*

Queste funzioni non sono sempre disponibili sulla tastiera. Nel caso esistano, sono chiamate in generale INST (INSERISCI) e DEL (CANCELLA).

Più soventemente si tratta di comandi all'editore, di speciali tasti funzione o associati a combinazioni di tasti di controllo o di un carattere.

c) *Altri comandi disponibili su alcuni sistemi*

Alcuni comandi riguardano l'annullamento dello schermo o il ritorno al punto iniziale dello schermo stesso. Se questi comandi sono disponibili sulla tastiera, si chiameranno allora ERASE (ANNULLA LO SCHERMO) o CLR. Il ritorno del cursore generalmente si chiama «HOME».

In alcuni sistemi (specialmente il sistema «APPLE»), si hanno le seguenti funzioni associate al tasto ESC (simboleggiato con E). Questo tasto deve essere combinato ad un altro carattere:

- E^E cancella dalla posizione del cursore fino alla fine della linea;
- F^E cancella lo schermo dalla posizione del cursore fino alla fine dello schermo;
- C^E cancella tutto lo schermo ed effettua il ritorno del cursore in alto a sinistra.

d) *I comandi associati al tasto di CONTROL*

Questi comandi sono più o meno standardizzati e si trovano su diversi sistemi. Essi associano il tasto CONTROL associato ad un'altra lettera sulle tastiere ASCII standard (standard americano utilizzato per la codifica dei caratteri).

Si ha specificatamente:

- CONTROL B simboleggiato con B^C seguito da ® che trasferisce il controllo al monitor BASIC e distrugge tutto il programma esistente (nei fatti equivale al tasto NEW);
- CONTROL G simboleggiato con G^C che fa emettere un «BIP» da un altoparlante incorporato o da una suoneria;
- CONTROL H (H^C): permette di effettuare un arretramento del cursore, cancellando i relativi caratteri;
- CONTROL V (V^C): fa avanzare il cursore verso destra, ricopiando i caratteri trovati;
- CONTROL J (J^C): mette in risalto una linea;
- CONTROL X (X^C): cancella la linea in cui si trova il cursore

CONCLUSIONE

In questo capitolo abbiamo preso in considerazione le caratteristiche generali del linguaggio e abbiamo presentato principalmente i comandi diretti del linguaggio e i comandi monitor suscettibili di essere disponibili sui sistemi BASIC. Bisogna insistere sul fatto che i comandi non fanno propriamente parte del linguaggio, che non esistono standard stretti, ma che le funzioni presentate, generalmente esistono in una forma analoga. Consigliamo dunque al principiante di verificare sul sistema utilizzato queste differenti funzioni e di provarle, prima di passare ai capitoli seguenti, che riguardano il linguaggio propriamente detto. In pratica, queste funzioni sono essenziali per guadagnare tempo, al momento della scrittura dei programmi o della loro modifica.

CAPITOLO 3

GLI ELEMENTI DI BASE DEL LINGUAGGIO

INTRODUZIONE

Un linguaggio, e specialmente un linguaggio di programma, è caratterizzato da un *alfabeto.*

Partendo da questo alfabeto, si possono costruire delle parole, utilizzando delle regole di costruzione che si chiamano regole *lessicali.* Le parole di un linguaggio sono riconosciute per il fatto che obbediscono a queste regole e sono inserite tra caratteri che le separano: il carattere bianco schematizzato con b̸ o □ o Δ viene generalmente utilizzato. Un certo numero di caratteri possono avere nel linguaggio un significato preciso: li si chiama allora *operatori.*

Allo stesso modo, un certo numero di parole hanno nel linguaggio un significato preciso: sono le *parole chiave.*

L'insieme ordinato delle parole, dei caratteri separatori e delle parole chiave costituiscono le *frasi* del linguaggio. In questo caso le si chiama anche *istruzioni.*

È evidente che tutte le frasi che si possono costituire non sono corrette. La loro costruzione obbedisce a certe regole che si chiamano *regole di sintassi* che costituiscono la *grammatica* del linguaggio.

Lo studio di un linguaggio di programmazione passa dunque dalla definizione arbitraria di un alfabeto e di regole lessicali. Per quanto concerne le regole sintattiche, queste non sono totalmente arbitrarie, in quanto devono permettere di definire degli algoritmi.

Tuttavia queste regole sintattiche devono essere scelte in modo tale da eliminare tutte le ambiguità nelle costruzioni di frasi corrette del linguaggio.

Un linguaggio di programmazione deve dunque essere non ambiguo, e, in pratica, ciò si traduce con la scelta di un certo tipo di grammatica detta *non contestuale.*

Tali grammatiche sono, d'altra parte, caratterizzate da certe proprietà che permettono di riconoscere facilmente una frase corretta da una che non lo è.

Ciò facilita la scrittura di programmi traduttori (compilatori o interpreti). Così le istruzioni possono essere tradotte senza ambiguità.

Ciò si traduce in possibili costrizioni al livello della programmazione: infatti la mancanza di un solo carattere o di una sola parola renderà scorretta una istruzione.

1. L'ALFABETO DEL LINGUAGGIO

L'alfabeto del linguaggio comprende tutti i caratteri utilizzabili nelle frasi di questo linguaggio. Ciò è anche chiamato l'elemento terminale del linguaggio.

Bisogna poi notare che un linguaggio di programmazione informatica utilizza un insieme di caratteri più esteso che l'alfabeto usuale.

Nel linguaggio BASIC, l'alfabeto comprende:

alfanumeriche
- le lettere maiuscole dalla A alla Z (26 caratteri);
- il carattere bianco: b̸ (o Δ o □) (1 carattere);
- le cifre decimali: da 0 a 9 (10 caratteri);
- i simboli speciali che comprendono gli operatori, i segni di punteggiatura e i separatori.
  + – / * ↑ operatori aritmetici
  + > < = operatori di relazione
  + % ( ) $ « separatori
  + . , : segni di punteggiatura

In tutto si ha un alfabeto di 53 caratteri.

Il carattere ® non fa parte dell'alfabeto propriamente detto, ma indica la fine di una linea di programma.

Nota. – Nel nuovo BASIC, sviluppato su microelaboratori, un certo numero di caratteri speciali è utilizzato per definire operazioni grafiche su schermo catodico o su televisore. Non daremo qui questi caratteri, poichè essi sono specificati da ciascun costruttore. Bisogna notare tuttavia che la serie di caratteri supplementari permette di sviluppare programmi di visualizzazione grafica o semigrafica che estendono le possibilità del linguaggio.

2. LE REGOLE DI FORMAZIONE DELLE PAROLE DEL LINGUAGGIO

Come si è già visto, le parole del linguaggio sono di tre tipi:
- gli *identificatori* scelti dal programmatore;
- le *costanti* definite dal programmatore;
- le *parole chiave* o parole riservate al linguaggio.

2-1. Gli identificatori delle variabili numeriche

Le regole di formazione degli identificatori sono, nel BASIC standardizzato, semplicissime: sono parole di due caratteri di cui il primo è una lettera e il secondo una cifra facoltativa.

Si tratta di una particolare limitazione agli «antichi» BASIC, che dà una possibilità di 286 nomi di differenti variabili nello stesso programma.

Gli identificatori possibili sono dunque:

A, A_0, A_1, ... A_9
B, B_0, B_1, ... B_9
.
.
.
.
Y, Y_0, Y_1, ... Y_9
Z, Z_0, Z_1, ... Z_9

Ciò rappresenta non di meno uno stock di nomi di variabili sufficiente, ma non è possibile dare dei nomi mnemonici alle variabili, e ciò costituisce un handicap per la comprensione dei programmi che non sono ben documentati.

Infatti, attualmente, sul BASIC esteso dei microelaboratori, il secondo carattere può essere un carattere alfanumerico, cosa che aumenta considerabilmente il numero delle possibili variabili. Sovente è possibile utilizzare degli identificatori più lunghi, ma soltanto i primi due caratteri sono utili.

Le variabili numeriche intere e reali

Nel BASIC non ci sono istruzioni di dichiarazione esplicite che permettono di definire le variabili intere o reali, non ci sono identificatori che siano esplicitamente considerati come interi.

Per definire una *variabile intera*, le si associa un identificatore con un carattere esplicito: %.

Così, l'identificatore N % è considerato come un nome di variabile intera.

In tutto il programma, una tale variabile non potrà contenere che dei valori interi positivi o negativi.

Tali variabili sono rappresentati internamente alla macchina con dei valori binari espressi da cifre binarie (i bits). Essendo una macchina caratterizzata da parole macchina che hanno una certa dimensione (8 bits, 16 bits, 32 bits), non sarà possibile rappresentare in una parola macchina dei numeri interi superiori al numero massimo che si può rappresentare con linguaggio binario in quella parola.

Così, se si hanno delle parole di 8 bits, si possono rappresentare i numeri positivi da 0 a $2^8 - 1$, cioè 255, oppure i numeri positivi e negativi da -128 a

+ 127. Per avere più dettagli su queste nozioni di calcolo binario, consigliamo il lettore di leggere l'appendice 1.

Generalmente se si hanno delle parole di *n* bits, i numeri interi che si possono rappresentare in una parola sono:

$$-2^{n-1} \leq N \leq 2^{n-1} - 1$$

Esempio:

Se la parola macchina è di 16 bits si ha

$$-2^{15} \leq N \leq 2^{15} - 1$$

cioè

$$-32768 \leq N \leq 32767$$

Le *variabili reali* sono definite con dei normali identificatori (per esempio N, X). Queste contengono dei valori detti reali, cioè dei numeri razionali comprendenti una parte intera ed una decimale. Per la maniera di rappresentazione interna di questi si rimanda all'appendice 1. Il campo di variazione per le variabili reali è ben più importante con la rappresentazione in virgola fluttuante (vedi sotto). Per alcuni sistemi BASIC tipici su microelaboratori, si ha un campo di variazione da 10^{-38} a 10^{38} con 9 cifre significative.

Nota. — Le forme di rappresentazione interna delle variabili reali e delle intere sono differenti. Per passare dall'una all'altra bisogna effettuare una conversione.

Restrizioni a proposito delle variabili intere.

Non è possibile utilizzare le variabili intere nelle istruzioni PER (FOR) e nelle istruzioni di definizione (DEF):

- le operazioni aritmetiche sono effettuate tra numeri reali e i valori interi sono dunque convertiti in reali prima di ogni calcolo;
- gli argomenti delle funzioni matematiche usuali (trigonometriche, esponenziali, logaritmiche ecc.) sono ugualmente convertiti in reali;
- il passaggio da un valore reale ad un valore intero implica una operazione che tronca la parte decimale.

Cosi se si vogliono eseguire le istruzioni:

```
10      N % = 4.556
20      PRINT N %
```

si ottiene:

```
   □    4
```

Ma bisogna fare attenzione, perchè se si ha:

```
10      N % = – 0.5
20      PRINT N %
```

si ottiene:

□ −1

Ed anche, se si ha:

```
10      N % = 0.98
20      PRINT N %
```

si avrà:

□ 0

In tutti i casi, si ottiene dunque la parte intera del numero come se si avesse utilizzato la funzione parte intera (INT).

L'interesse delle variabili intere è dovuto essenzialmente al guadagno di spazio in memoria, ottenuto in rapporto ai numeri reali. Queste variabili interessano in particolare quando si vogliono definire degli indici di elenchi o di tabelle, poichè questi sono necessariamente valori interi.

Gli identificatori delle variabili stringhe di caratteri

Alcuni BASIC limitano l'identificatore ad una sola lettera, ma, per la maggior parte dei BASIC, si hanno le stesse regole viste per gli identificatori delle variabili numeriche.

Basta semplicemente aggiungere il carattere $ dopo l'identificatore.

Esempio:

A $, B $, ... Z $ ....
A 1 $............................ Z 9 $.........

rappresentano gli identificatori delle variabili stringhe di caratteri.

Con il BASIC si ha quindi la possibilità di avere da 26 a 286 identificatori.

I BASIC estesi accettano tutti gli identificatori che cominciano con una lettera e che sono seguiti da un carattere alfanumerico.

2-2. Le costanti

Le costanti sono, allo stesso modo, di due tipi:

- le costanti numeriche;
- le costanti stringhe di caratteri

a) *Le costanti numeriche*

Sono numeri che possono essere interi o frazionari e che hanno il segno + o −.

Se il segno è positivo può essere omesso. I numeri decimali sono rappresentati da una parte intera e una decimale, separate da un punto e non da una virgola.

Esempio:

146 è una costante intera.
84 è una costante intera.
153.4 è una costante frazionaria decimale ancora chiamata reale.
3.14 è una costante frazionaria decimale ancora chiamata reale, ma il numero di decimali è limitato.

Nota. — Quando la parte intera è nulla, si può omettere lo 0. Quindi la costante 0.5 può essere scritta: .5.

La rappresentazione in virgola fluttuante

Internamente le costanti reali sono codificate con l'aiuto di una rappresentazione detta in virgola fluttuante.

Il principio è semplice, in quanto è basato sulla rappresentazione esponenziale.

In effetti, un numero come: 454.87 può essere rappresentato sotto differenti forme, come:

$$4.5487 \times 10^2$$
$$45487 \times 10^{-2}$$
$$0.45487 \times 10^3$$

Si vede dunque che, moltiplicando per una potenza di 10 positiva o negativa, si può far spaziare il punto all'interno delle cifre significative, da cui deriva il nome di virgola (o punto) fluttuante. In questa rappresentazione le cifre significative costituiscono ciò che si chiama la *mantissa*. Conoscendo le potenze del 10 associate, si può risalire al numero.

Si dice poi forma *normalizzata* quella per cui la mantissa è compresa tra 0.1 e 0.99... Nell'esempio precedente, la forma normalizzata è 0.45487×10^3. La forma normalizzata è utilizzata alcune volte in uscita dagli interpretatori BASIC. In questo caso, per rappresentare un numero fluttuante, si utilizzerà il carattere «E» come separatore tra mantissa ed esponente.

Esempio:

0.314E1	= 3.14
.14768E3	= 147.68
.5E0	= 0.5
.25E – 2	= 0.0025

Nota. — Questa notazione, utile per problemi scientifici, non lo è altrettanto per problemi di gestione, in quanto è inaccettabile per stampare documenti come fatture, note bancarie, ecc.

Bisogna rilevare che la maggior parte dei BASIC dispongono di programmi di conversione dalla forma fluttuante interna alla forma decimale usuale.

b) *Le costanti stringhe di caratteri*

Abbiamo già visto degli esempi riguardanti il caso che si vogliano scrivere dei testi.

Le regole concernenti queste costanti sono quelle per cui esse sono incorniciate da caratteri speciali: " (virgolette).

Esempio:

"GIULIO", "PARIGI"
"QUESTA È UNA STRINGA"

La sola regola da rispettare è la lunghezza della serie di caratteri: questa è fissata in un massimo di 225 caratteri, comprendendo gli spazi bianchi. Nel BASIC esistono delle istruzioni di manipolazione dei caratteri assai complete: verranno presentate in un successivo paragrafo.

Esercizi

1. *I seguenti nomi sono corretti nel BASIC esteso?*
   N1, JOJO, PA, RS, LUNA, 1K, B100, 44A, 0SS117, M + 1
2. *Sono corrette le seguenti costanti?*
   15 24,15 72.54 3^2 − 14 ± 31.1 .55
   0.68E + 2 1.24, E − 4 2/3/79.
3. *Scrivere qualche costante stringa di caratteri. Si possono utilizzare parole chiave all'interno delle stringhe di caratteri?*

Risposte

1. SI − NO − SI − SI − NO − NO − NO − NO − NO − NO
2. SI − NO − SI − NO − SI − NO − SI − SI − NO − NO
3. "BASTA IMMAGINARLE".
   Sì, si possono utilizzare delle parole chiave internamente alle stringhe di caratteri.

3. LE PAROLE CHIAVE DEL LINGUAGGIO

Le parole chiave, o parole riservate, del linguaggio BASIC sono di origine inglese. Noi le daremo in italiano con la traduzione in inglese.

Le abbiamo classificate per categorie e in ordine alfabetico.

Parole chiave di dichiarazione

DIM	dichiara le dimensioni della tabella
DATI	DATA

Parole chiave condizionali

SE ALLORA	IF THEN

Parole chiave iterative

CON IL PASSO	FOR TO STEP
SEGUENTE	NEXT

Parole chiave imperative

REM	Nota
SIA	LET
ENTRA	INPUT
STAMPA	PRINT
DEF FN	Definizione di funzione
LEGGI	READ
RILEGGI	RESTORE

Parole chiave di controllo

SU	ON
TORNA	RETURN
VAI A	GO TO
SOTPROG	GOSUB

Parole chiave operatrici

In alcuni BASIC esistono operatori logici come i seguenti:

E	END
O	OR
NON	NOT

Parole chiave di funzioni matematiche

FN	Funzione definita dal programmatore
ABS	Valore assoluto
EXP	Esponenziale
LOG	Logaritmo neperiano
DLOG	Logaritmo decimale
COS	Coseno
SIN	Seno
TAN	Tangente
ATN	Arcotangente
RAD (SQR)	Radice quadra
INT	Valore intero
RND	Generatore aleatorio
SGN	Segno
DEG	Conversione radianti gradi
RAD	Conversione gradi radianti

Parole chiave di funzione di manipolazione delle stringhe di caratteri

ASC		Conversione ASCII decimale
CAR$	CHR$	Conversione decimale ASCII
LUN	LEN	Lunghezza di una stringa di caratteri
VAL		Conversione stringa di caratteri-numeri
CHA$	STR$	Stringa di caratteri corrispondenti ad un numero
MIL$	MID$	Estrazione di una sotto stringa di caratteri
DESTRA$	RIGHT$	Estrazione di una stringa a partire dalla destra
SINIST$	LEFT$	Estrazione di una stringa a partire dalla sinistra

Questa lista di parole rappresenta lo standard BASIC più esteso.
In alcuni BASIC su microelaboratori si possono anche trovare le seguenti parole chiave

PEEK	Lettura di un bit (parola memoria)
POKE	Scrittura di bit (parola memoria)
CALL	Richiamo di un sottoprogramma in linguaggio
o USEP	macchina

Esistono poi parole chiave più specifiche per i BASIC che permettono di fare grafici.

Su alcuni BASIC evoluti esistono poi parole chiave che permettono manipolazioni su matrici (MAT, ZER, TRN, INV, IDN, CON).

Infine, esistono istruzioni di manipolazione di «files» (schedari) sui BASIC estesi e sui microelaboratori. Ne parleremo in un prossimo capitolo.

4. LE REGOLE DI PROGRAMMAZIONE NEL LINGUAGGIO BASIC

Fino ad ora, abbiamo dato degli esempi e definito gli elementi di base del linguaggio: l'alfabeto, gli identificatori, le costanti, le parole chiave..., ecc. Bisogna ora imparare a servirsene e a creare delle frasi o delle istruzioni che siano sintatticamente corrette, cioè che rispettino le regole fondamentali del linguaggio. In programmazione, queste regole sono precise, e, addirittura, per il neofita, rigide. È dunque importante conoscerle bene e ben utilizzarle.

La seconda tappa consiste nel creare un programma che abbia senso! Ci si mette ad un livello che non dipende più né dalla macchina, né dal linguaggio, ma dall'intelligenza dell'utente. Si è quindi ad un livello che si chiama semantico.

Gli esempi che daremo sono stati scelti per il loro valore pedagogico e sono semplici, in modo da mostrare cosa è possibile fare con il linguaggio.

Questi esempi devono permettere all'utente di acquisire un certo numero di riflessi, ma è evidente che solo la *pratica* del linguaggio permette di diventare un maestro.

È dunque importante non fermarsi alla sola lettura dei programmi, o alla loro ricopiatura su una macchina, ma è necessario esaminare altri problemi e tentare di risolverli da soli. Nel seguito, speriamo di permettere al lettore di apprendere tutti i concetti utili alla programmazione e di fornirgli tutti i mezzi corrispondenti al linguaggio studiato.

Nel seguito del capitolo, vedremo in dettaglio i differenti tipi di istruzioni: le istruzioni aritmetiche, le istruzioni di test, le istruzioni di iterazione, quelle di entrata - uscita, quelle di manipolazione di catene di caratteri.

4-1. I differenti tipi di istruzione in BASIC

Tralasciando l'istruzione REM, che rappresenta solo un commento, si distinguono cinque grandi categorie di istruzioni:

- le istruzioni di dichiarazione (DIM, DATA);
- le istruzioni di assegnazione (LET), tra le quali si trovano le istruzioni aritmetiche e le istruzioni di manipolazione di stringhe;

- le istruzioni di test e di rottura di sequenza (SE... ALLORA... VAI A) (IF... THEN... GO TO...);
- le istruzioni di anello di iterazione (PER) (FOR);
- le istruzioni di entrata-uscita (INPUT, PRINT, READ, WRITE).

Struttura di una istruzione

Nel BASIC standard, è d'uso accettare solo una istruzione per linea, la composizione della quale è di 80 caratteri. Una istruzione è quindi composta da un numero, seguito dal testo dell'istruzione: questo numero è anche chiamato *etichetta*.

Esempio:

```
10      LET I = 1
20      INPUT N
```

Questa regola, attualmente, non è più generale, in quanto, nella maggior parte dei BASIC estesi, è possibile definire più istruzioni sulla stessa linea, a condizione di separare ogni istruzione con un carattere delimitante, generalmente rappresentato dai due punti : .

In questo caso non è possibile etichettare o numerare ogni singola istruzione, non potremo più riferirci ad essa, quindi, nel resto del programma.

Esempio:

```
10      LET I = 1 ; INPUT N
```

Qui, per esempio, l'istruzione INPUT N non può più essere di riferimento nel resto del programma. Bisogna dunque essere prudenti quando si definiscono linee con più istruzioni. Se si desidera che il programma sia compatibile con diversi BASIC, è meglio non utilizzare questa possibilità.

Facciamo notare tuttavia che, sulla maggior parte dei microelaboratori attuali, gli interpreti sono stati realizzati dalla stessa società (MICROSOFT) e che, quindi, le compatibilità sono assai buone.

Il problema si pone quando si passa da un BASIC esteso a dei BASIC standard o un po' obsoleti.

In questo senso, si deve rilevare che il BASIC dei microelaboratori è spesso più sofisticato di quello che si può trovare su macchine anche più potenti.

Per terminare un programma: Istruzione FINE e STOP

Prima di studiare le differenti istruzioni, dobbiamo sapere come terminare un programma. C'è l'istruzione FINE (STOP): questa deve essere dunque l'ultima istruzione del programma. Quando l'interprete incontra questa istruzione, il programma si arresta e si ritorna al monitor.

Tuttavia, se si vuole poter distinguere la fine dall'arresto dell'esecuzione del programma, arresto che può essere ripetuto più volte nel programma, si utilizzerà l'istruzione STOP. Questa non è necessariamente l'ultima istruzione del programma, ma, quando l'interprete la incontra, si ha l'arresto del programma e il ritorno al monitor. Si possono dunque avere più istruzioni STOP nello stesso programma. È comunque possibile far ripartire un programma dopo una istruzione STOP (con il comando CONT).

4-2. Istruzioni di calcolo aritmetico

Abbiamo visto nei capitoli precedenti dei semplici esempi di calcolo aritmetico. In programmazione, le istruzioni aritmetiche permettono di precisare dei calcoli che, più soventemente vengono chiamati algebrici. Nel BASIC standard, queste istruzioni cominciano con la parola chiave LET (SIA), seguita da una istruzione aritmetica.

Tuttavia, sulla maggior parte dei BASIC attuali, la parola chiave LET è *facoltativa*; in questi casi si può scrivere direttamente l'istruzione aritmetica dopo il numero di identificazione dell'istruzione.

Così:

```
10      LET A = B + C
```

equivale a

```
10      A = B + C
```

LE ISTRUZIONI ARITMETICHE O ISTRUZIONI DI ASSEGNAZIONE

Una tale istruzione è caratterizzata da una parte sinistra che deve contenere un *identificatore* di variabili e da una parte destra che contiene una espressione aritmetica. I due membri sono separati dal segno = che, per ora, noi abbiamo considerato segno di uguaglianza. Infatti, in programmazione, que-

sto segno ha un altro significato e lo si chiama più soventemente segno di assegnazione (in altri linguaggi di programmazione lo si nota con: = o ←.).

In effetti, questo segno indica che la macchina deve effettuare il calcolo dell'espressione che si trova nella parte destra dell'istruzione e che il risultato ottenuto deve essere assegnato e memorizzato nella variabile definita dalla parte sinistra dell'istruzione.

Così, in programmazione, una istruzione del genere

10 X = X + 2

è totalmente corretta. Non deve cioè essere considerata né una uguaglianza, né una equazione. Indica infatti che si effettua la somma della variabile X e della costante 2 e che il risultato deve essere assegnato alla parte sinistra, dove già si trova la variabile X che definisce la parte sinistra dell'istruzione. Così, per esempio, se il contenuto di X era 5 prima dell'esecuzione dell'istruzione 10, il risultato finale sarà 5 + 2, cioè 7, e il nuovo valore di X sarà, dopo l'esecuzione dell'istruzione 10, 7.

Si può dunque dire che l'istruzione di assegnazione permette di modificare in maniera dinamica il contenuto di una variabile. Ciò può essere utilizzato negli algoritmi di tipo iterativi basati su formule ripetitive.

Esempio:

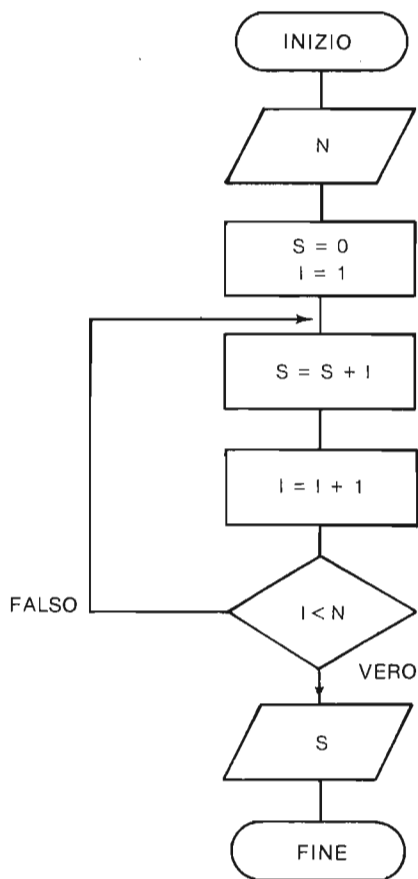
Si debba calcolare la somma di N numeri interi.

L'algoritmo può essere specificato con una formula iterativa: in effetti, supponendo conosciuta la somma degli $i - 1$ numeri, e sia S_{i-1}, si ottiene la somma degli i numeri sommando i a S_{i-1}.

$$\text{Sia: } S_i = S_{i-1} + i$$

L'algoritmo è allora il seguente:

- I_1 LEGGI N
- I_2 S = 0
- I_3 I = 1
- I_4 S = S + 1
- I_5 I = I + 1
- I_6 SE I > N allora stampa S STOP
- I_7 Altrimenti vai a I_4

Lo schema a blocchi corrispondente è

La programmazione diretta da questo schema a blocchi dà nel linguaggio BASIC:

```
10    ENTRA N                  10    INPUT N
20    S = 0                    20    S = 0
30    I = 1                    30    I = 1
40    S = S + 1                40    S = S + 1
50    I = I + 1                50    I = I + 1
60    SE I = N ALLORA 40       60    IF I = N THEN 40
70    STAMPA S                 70    PRINT S
80    FINE                     80    END
```

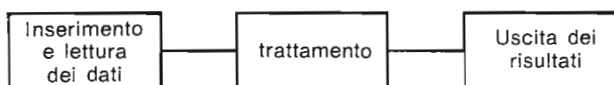
Nei fatti, questo modo di programmare il problema sopra visto non è né

migliore, né il più conciso; si sarebbe poi potuto utilizzare l'istruzione PER (FOR). Lo rivedremo nel paragrafo corrispondente a questa istruzione. Per ora ragioniamo ancora sull'esempio precedente:

Il programma comprende tre parti:

- una parte statica, corrispondente all'iniziare delle variabili e alla lettura dei dati (istruzioni 10, 20, 30);
- una parte dinamica corrispondente all'accumulazione dei risultati intermedi nelle variabili S e I (istruzioni 40, 50, 60), questa parte corrisponde alla formula iterativa associata all'algoritmo;
- una parte terminale, statica, che stampa i risultati.

Questa struttura è presente, in generale, in tutti gli algoritmi:

- *Prima parte:* inizializzazione delle variabili e/o lettura dati.
- *Seconda parte:* è dinamica e corrisponde al lavoro da effettuare.
- *Terza parte:* è l'ultima e corrisponde all'uscita dei risultati e all'arresto dell'algoritmo.

I DIFFERENTI TIPI DI ESPRESSIONI ARITMETICHE

Una istruzione aritmetica ha la forma:

SIA V = Espressione aritmetica

dove V è una variabile e = è il segno di assegnazione.
La parola chiave SIA (LET) è facoltativa nella maggior parte dei BASIC. In seguito non la utilizzeremo più.

Una espressione aritmetica semplice è caratterizzata da:

- una serie di nomi di variabili o di costanti separati da operatori aritmetici.

Gli *operatori aritmetici* sono in numero di cinque:

+ addizione
— sottrazione
* moltiplicazione
/ divisione
↑ elevamento a potenza.

Per tradurre una qualsiasi espressione matematica, basta linealizzarla, utilizzando gli operatori sopra definiti. Bisogna tuttavia tener conto di tutte le operazioni, anche se esse non sono esplicitate nell'espressione iniziale.

Esempio:

L'espressione $ab + cd$ si traduce con:

A * B + C * D

e, allo stesso modo,

$$b^2 - 4ac$$

si traduce con

B ↑ 2 – 4 * A * C

Regole di precedenza o gerarchia degli operatori

Consideriamo l'espressione

A * B / C * D

in assenza di regole di precedenza essa può essere interpretata in diversi modi

$$a \times b / c \times d \text{ o } \frac{a \times b}{c} \times d$$

In pratica, nel BASIC, le espressioni non possono ricorrere ad artifici tipografici usuali in matematica. Bisogna dunque definire un certo numero di regole che diano, di una certa espressione, un'unica interpretazione. Si è definita allora una gerarchia di operatori aritmetici chiamati *precedenza degli operatori.*

Nel contempo, si sono definite le regole concernenti le espressioni aritmetiche *semplici*, cioè quelle che non fanno intervenire parentesi o funzioni:

Regola 1. — Una espressione aritmetica semplice è valutata tenendo conto della operazione di elevamento a potenza (↑).

Poi, andando da sinistra verso destra, si effettuano le operazioni di moltiplicazione (*) o di divisione (/).

Infine, sempre andando da sinistra verso destra, le operazioni di addizione (+) e di sottrazione (—).

Si ha dunque la seguente tabella:

- elevamento a potenza ↑ — priorità 1
- moltiplicazione * divisione / — priorità 2
- addizione + sottrazione — — priorità 3

Esempio:

Riprendiamo l'espressione vista prima:

A * B / C * D

sarà valutata nel modo seguente:

- tutte le operazioni hanno priorità 2;
- si effettuano le operazioni andando da sinistra verso destra,

. $A \times B$

. $\frac{A \times B}{C}$

. $\frac{A \times B}{C} \times D$

L'espressione matematica corrispondente non è

$$\frac{A \times B \times D}{C} \quad \text{e non} \quad \frac{A \times B}{C \times D}$$

Vediamo ora un'espressione contenente tutti gli operatori:

A / B + C – D / E * F ↑ 2.

Questa espressione sarà valutata effettuando dapprima l'operazione

F^2 (F ↑ 2)

poi $\frac{A}{B}$ e $\frac{D}{E} \times F^2$

infine: $\frac{A}{B} + C - \frac{D}{E} \times F^2$

Introduzione delle parentesi

Alcune espressioni matematiche più complesse necessitano dell'introduzione delle parentesi. In pratica, in programmazione, non è possibile definire delle espressioni complesse con degli artifici tipografici che utilizzano «impalcature» matematiche del tipo:

$$\frac{a + \frac{b}{c}}{c + \frac{d + e}{k}}$$

Perciò bisogna introdurre le parentesi. Qui il numeratore è equivalente a: A + B / C.

Ed il denominatore è equivalente a: C + D / K + E / K.

In effetti, l'espressione C + D + E / K non è la corretta rappresentazione di $c + \frac{d + e}{k}$.

Di contro, se si introducono le parentesi, si può scrivere:

C + (D + E) / K

Allo stesso modo, per indicare la divisione dell'espressione numeratore per l'espressione denominatore, le parentesi sono opportune. Così l'espressione finale corretta è:

(A + B / C) / (C + (D + E) / K)

Si deduce quindi la seguente regola:

Regola 2. — Le espressioni tra parentesi sono valutate con una priorità superiore a tutte le altre operazioni.

La parentesi può essere dunque considerata di priorità 0.

Nota. — Certe espressioni possono essere programmate in diversi modi. Così l'espressione $\frac{a \times b}{c \times d}$ può essere programmata in tre modi:

— il primo modo è

(A * B) / (C * D)

— il secondo è

A * B / (C * D)

Questa espressione è corretta in quanto si effettua prima l'operazione (C × D), poi A × B che verrà diviso per C × D.

– il terzo modo è

A * B / C / D

In questo caso non ci sono le parentesi, tuttavia il risultato è corretto. Si effettua infatti, prima A × B, questo viene diviso per C, avremo cioè $\frac{A \times B}{C}$; questo ultimo risultato verrà ancora diviso per D, in modo tale da avere $\frac{A \times B}{C \times D}$.

Si vede dunque che la programmazione di una formula con frazioni non necessita sempre di parentesi. Si deve tuttavia rilevare che è preferibile l'uso di inutili parentesi ad una forma scorretta. Così, per un principiante, è preferibile l'introduzione di parentesi, poiché queste danno più certezze sulla validità di una espressione.

Le funzioni matematiche standard del BASIC

Alcune espressioni matematiche utilizzano funzioni matematiche standard: radici quadre, esponenziali, logaritmi, funzioni trigonometriche, ecc...

In questo caso, una funzione è caratterizzata dal *nome* della funzione, che è una parola chiave del linguaggio, seguito, tra parentesi, da una espressione aritmetica. Ciò suppone evidentemente che l'espressione tra parentesi sia definibile, corrisponda cioè al dominio di definizione della funzione: così, se si utilizza la funzione radice quadra, si suppone che l'espressione associata sia sempre positiva o nulla. Certamente dipende dall'abilità del programmatore assicurarsi che ciò sia sempre verificato, altrimenti il programma potrà essere sintatticamente corretto, ma sarà causa di errori nell'esecuzione. L'elenco delle funzioni matematiche usuali nel BASIC viene dato nella tabella 2.

TABELLA 2: FUNZIONI MATEMATICHE STANDARD IN BASIC

ABS (X)	Valore assoluto di X
ATN (X)	Arcotangente di X
COS (X)	Coseno di X
EXP (X)	Esponenziale di X
INT (X)	Parte intera di X
LOG (X)	Logaritmo neperiano di X
RND (X)	Valore aleatorio tra 0 e 1
SGN (X)	Segno di X (+1 se pos, 0 se nullo, −1 se negat.)

SIN (X) Seno di X
SQR (X) Radice quadra di X («square root»)
TAN (X) Tangente di X

In alcuni BASIC si possono trovare altre funzioni come:

COT (X) Cotangente di X
DLOG (X) Logaritmo decimale di X

Tuttavia, in ogni caso, le funzioni supplementari esistenti in alcuni elaboratori possono essere calcolate partendo dalle funzioni standard.

Esempio:

COT (X) = 1 / TAN (X)
DLOG (X) = LOG (X) / LOG (10)

Non è dunque necessario disporre di queste funzioni per poterle definire. L'elenco di queste funzioni derivate verrà dato più avanti.

Introduzione delle funzioni standard in una espressione aritmetica

Gli argomenti delle funzioni matematiche possono essere delle espressioni aritmetiche tra parentesi. Ci si ritrova dunque al livello di priorità 0 per la valutazione delle espressioni corrispondenti, ma, in più, quando questa espressione sarà stata valutata, la funzione sarà simultaneamente calcolata.

Si può dunque enunciare la seguente regola:

Regola 3. — In una espressione aritmetica le funzioni matematiche sono valutate con priorità 0, equivalente alle espressioni tra parentesi.

Esempio:

Si abbia l'espressione

$$\frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Questa può essere programmata nella forma:
(—B + SQR (B ↑ 2 — 4 * A * C) / (2 * A)

Innanzitutto si effettuano le espressioni tra parentesi e le funzioni, cioè:
SQR (B ↑ 2 — 4 * A * C)

poi

$$- B + SQR (B \uparrow 2 - 4 * A * C)$$

poi

$$2 * A$$

ed infine si divide la prima espressione per la seconda.

L'operatore monadico negazione

Si abbia l'espressione

$$R = C \times \frac{I}{1 - (1 + I)^{-N}}$$

che rappresenta il calcolo di un rimborso R di un capitale C rimborsato dopo N anni all'interesse dell'I%.

Questa espressione può essere così programmata:

R = C * I / (1 − (1 + I) ↑ (− N))

Nel BASIC esteso la si può ugualmente scrivere cosi:

R = C * I / (1 − (1 + I) ↑ − N)

In questa espressione si vede l'introduzione del segno — per rappresentare l'operazione di negazione (in questo caso — N).

Si può quindi considerare l'operatore negazione, qui rappresentato dal simbolo —, come un operatore individuale agente solo su un operando (si dice anche operatore unitario o monadico). Questo operatore avrà una priorità superiore agli operatori che fanno intervenire due operandi (detti anche diadici) come * / oppure + —. Quindi l'operatore *monadico* può essere considerato come avente priorità 1 *bis* tra l'esponenziale e la moltiplicazione e la divisione.

Esempio

L'espressione — A * B + C è equivalente a:

(− A) * B + C

che ci permette di effettuare subito l'operazione di negazione su A, prima di moltiplicare per B ed infine aggiungere il valore C.

RIASSUNTO SULLE ISTRUZIONI ARITMETICHE

Abbiamo visto che una istruzione aritmetica incomincia con la parola

LET (in genere facoltativa), seguita da una istruzione del tipo:

(LET) nome della variabile = espressione aritmetica

Le regole sintattiche concernenti le espressioni aritmetiche sono le seguenti:

Regola 1. — Le priorità degli operatori nella valutazione delle espressioni aritmetiche sono le seguenti:

1. — Parentesi o funzioni matematiche. Le parentesi e le funzioni più interne sono effettuate subito.
2. — L'elevamento a potenza (↑).
3. — La negazione (—) operazione monadica.
4. — La moltiplicazione (*) e la divisione (/) nell'ordine in cui sono, andando da sinistra a destra.
5. — L'addizione (+) e la sottrazione (—) nell'ordine in cui sono, andando da sinistra a destra.

Se ora, utilizziamo tali espressioni in un programma, bisognerà tener conto di un certo numero di regole dette semantiche; di queste, le due principali riguardano la conoscenza preliminare dei valori delle variabili e l'omogeneità delle espressioni.

Regola 2. — Tutte le variabili utilizzate in un'espressione aritmetica devono avere un valore al momento dell'esecuzione dell'istruzione. *In generale, il valore preso per difetto è 0.*

Regola 3. — Tutte le variabili o le costanti di una istruzione aritmetica devono essere variabili numeriche, ad esclusione delle variabili o costanti del tipo stringhe di caratteri.

Esercizi sulle istruzioni aritmetiche

1. *Tradurre in BASIC le seguenti espressioni:*

$$4x + 5y, \frac{x + y}{z}, (a + b)^2, e^{-(x + y)}, \frac{n(n + 1)}{2}$$

$$\text{Log}\,\frac{a + b}{c + d}, \sqrt{a^2 + b^2}, \pi r^2, \cos\varphi$$

2. *Scrivere le espressioni matematiche corrispondenti alle seguenti espressioni BASIC e valutarle per valori di:*
A = 2, B = 3, C = 4:

A* B + C / A (A / 2 + B / 3 + C / 4) / 3
A ↑ 2 + B ↑ 2 + 2* A* B (A + B) ↑ 2
A + SQR (B^2 + C^2) / 5 * A C * (1 + B / 100) ↑ A

3. *Si abbia la serie delle seguenti istruzioni aritmetiche: calcolare il valore delle tre variabili A, B, C dopo l'esecuzione di ciascuna delle istruzioni, per* A = 2, B = 3, C = 4.

C = A + B + C / A
B = B + C
A = A * B + C * A ↑ 3
C = A − B * C
B = (A − 4) / C
A = A − B * C
C = B − A
B = B / 2

SOLUZIONE DEGLI ESERCIZI

1. 4 * X + 5 * Y
   N * (N + 1) / 2
   EXP (−(X + Y))
   SQR (A↑2 + B↑2)
   COS (FI)
   (X + Y) / Z
   (A + B)↑2
   LOG ((A + B) / (C + D))
   PI * R↑2
   o π R ↑ 2 su quei microelaboratori aventi il tasto π sulla tastiera.

2. $ab + \frac{c}{a}$ valore : 8

   $\frac{\frac{a}{2} + \frac{b}{3} + \frac{c}{4}}{3}$ valore : 1

   $a^2 + b^2 + 2ab$ valore : 25

   $(a + b)^2$ valore : 25

   $a + \frac{\sqrt{b^2 + c^2}}{5} \times a$

   $c(1 + \frac{b}{100})^a$ valore : 4.2436

3.

	A	B	C
	2	3	4
$c = a + b + c/a$	2	3	7
$b = b + c$	2	10	7
$a = a * b + c * (a)^3$	76	10	7
$c = a - b * c$	76	10	6
$b = \frac{a - 4}{c}$	76	12	6
$a = a - bc$	4	12	6
$c = b - a$	4	12	8
$b = b/2$	4	6	8

Nota. — Per chi dispone di una macchina che lavora in BASIC, basta riprendere le istruzioni precedenti ed inserire all'inizio una istruzione INPUT A, B, C, e, dopo ogni istruzione, una istruzione PRINT A, B, C.

Esercizi

Convertire in BASIC le seguenti espressioni:

a) $2i + 4j$
b) $2a + 3b$
c) $a^2 - 4b$
d) $2(a + b)$
e) $(a + 2b + 3c)^2$
f) $\left(\frac{a}{b}\right)^2$
g) $\frac{a + b}{c + d}$
h) $\frac{a}{b} + \frac{c}{d}$
i) $a^2 - 4ab$
j) $\frac{x}{2!} + \frac{x^2}{4!}$

4-3. Le istruzioni di manipolazione delle stringhe di caratteri

Come per il calcolo aritmetico, esistono quelle che si chiamano costanti stringhe di caratteri e di variabili del tipo stringhe di caratteri.

Le costanti stringhe di caratteri

Abbiamo visto che queste costanti sono delimitate da due segni identici (uno all'inizio e uno alla fine), che sono le virgolette: ". La costante è rappresentata dall'insieme dei caratteri tra virgolette.

Esempi:

– "CIAO" è una costante stringa di caratteri che rappresenta il messaggio CIAO.
– "QUESTA È UNA LUNGA STRINGA DI CARATTERI" è una stringa di caratteri comprendente più parole separate da spazi bianchi.
– "8 DICEMBRE 1947" è una stringa di caratteri indipendentemente dal fatto che contiene delle cifre. In questo caso esse saranno considerate come dei caratteri.

STAMPA DIRETTA DI STRINGHE DI CARATTERI

Nello stesso modo in cui è possibile utilizzare il BASIC per fare dei calcoli diretti, è possibile utilizzarlo per stampare caratteri.

Così, se si scrive:

PRINT "CIAO" ®

la risposta è

CIAO

Tutto ciò presenta sicuramente un interesse limitato nella misura in cui non si ha alcun lavoro sulla stringa di caratteri. Bisogna tuttavia notare che, se si dispone di una stampante, si può anche stampare una serie di messaggi per un destinatario X, senza saper programmare.

L'OPERAZIONE DI CONCATENAMENTO

Si tratta di una operazione importantissima, semplice e necessaria per la manipolazione delle catene di caratteri.

Definizione. — Il *concatenamento* di due stringhe di caratteri consiste nell'unire le due stringhe iniziali, aggiungendo i caratteri della seconda stringa immediatamente dopo quelli della prima.

Esempio:

- Il concatenamento di "SOTTO" e di "SOPRA" diviene "SOTTOSOPRA"
- Il concatenamento di "BUONA" e di "SERA" diviene "BUONASERA"
- Il concatenamento del prefisso "ANTI" con "COSTITUZIONAL" e con il suffisso "MENTE" dà "ANTICOSTITUZIONALMENTE".

Quindi il concatenamento è una operazione che permette di ottenere delle stringhe di caratteri, per costituire delle parole a partire da lettere o da frasi a partire da parole. È quindi una operazione che è alla base di tutto il linguaggio.

Notazione del concatenamento in BASIC

Poiché si tratta di una operazione di somma, in BASIC, l'operatore del concatenamento è +.

Non esiste ambiguità con il segno + aritmetico, nella misura in cui non si hanno miscugli di tipi di variabili numeriche e di stringhe di caratteri.

Esempio:

- "GIAN" + "PIERO" dà "GIAN PIERO"

Si possono ugualmente far eseguire degli ordini di stampa diretta, utilizzando l'operazione di concatenamento.

Esempio:

– PRINT "PIETRO" + "E" + "PAOLO" ®□ PIETRO E PAOLO
– PRINT "BUONGIORNO" + "SIGNORA" ® □ BUONGIORNO SIGNORA

LA RAPPRESENTAZIONE DEI CARATTERI IN MACCHINA

Si è visto nel primo capitolo che, per motivi interni, una macchina manipola le informazioni sotto forma di *parole* macchina che possono contenere solo informazioni binarie (serie di 0 o 1). Ciò presuppone quindi che tutti i caratteri siano rappresentati da un codice binario interno.

I codici attualmente più utilizzati sono codici ad *otto bits* (cosa che permette di rappresentare fino a 2^8, cioè 256, caratteri differenti).

Il codice più diffuso sulle piccole macchine, ed, in particolare, sui microelaboratori, è il codice ASCII («American Standard Communication Informations Interchange»), di cui viene data qui sotto la tabella di corrispondenza.

TABELLA DEI CODICI ASCII

Caratteri o controllo	ASCII Esadecimali	Caratteri speciali e cifre	ASCII Esadecimali	Caratteri maiuscoli	ASCII Esadecimali	Caratteri minuscoli	ASCII Esadecimali
NUL	00	SP (spazio)	20	@	40	\	60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	S	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	/	27	G	47	g	67
BS	08	(	28	H	48	h	68
HT	09	)	29	I	49	i	69
LF	0A		2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	–	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F
DLE	10	0	30	P	50	p	70
DCA (X-ON)	11	1	31	Q	51	q	71
DC2 (TAPE)	12	2	32	R	52	r	72
DC3 (X-OFF)	13	3	33	S	53	s	73
DC4 (TAPE)	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[	5B	{	7B
FS	1C	<	3C	\	5C	¦	7C
GS	1D	=	3D	]	5D	}(ALT MODE)	7D
RS	1E	>	3E	Δ(↑)	5E		7E
US	1F	?	3F	– (←)	5F	DEL	7F

È interessante notare sulla tabella dei codici ASCII che, se si guardano i codici associati alle lettere, ci sono dei numeri crescenti andando dalla A alla Z. Il carattere bianco, poi, possiede un codice inferiore a quello dell'insieme delle lettere dell'alfabeto. Queste precisazioni fanno sì che, quando si vorrà classificare delle stringhe di caratteri in ordine alfabetico, per esempio, si potrà facilmente farlo sulle stringhe di caratteri alfabetici, ma bisognerà diffidare dei caratteri bianchi intercalati. Torneremo su questo problema più avanti.

Le variabili stringhe di caratteri

Abbiamo visto all'inizio del capitolo che i nomi di queste variabili erano composte da una lettera seguita o no da un carattere alfanumerico e terminavano con il carattere $.

Semplice espressione della stringa di caratteri

Una espressione semplice di stringa di caratteri è una espressione comprendente il concatenamento di una o più variabili o costanti stringhe di caratteri.

Esempio:
- "LE" + N$
- A$ + B$ + "CATENA 3"
- NO$ + PR$ + AD$

Istruzioni di assegnazione delle stringhe di caratteri

Una istruzione di assegnazione ha la stessa struttura che le istruzioni aritmetiche:

Nome della variabile stringa = espressione stringa

L'espressione del membro a destra viene subito trattata, e il risultato è assegnato alla variabile situata nella parte sinistra. Si può dunque anche avere lo stesso nome della variabile alla sinistra e alla destra del segno di assegnazione (=).

Esempio:

Siano NO$ e CO$ due variabili rappresentanti le stringhe di caratteri nome e cognome.
Si può definire una variabile IDS del tipo:

```
10      (LET) ID$ = NO$ + CO$
```

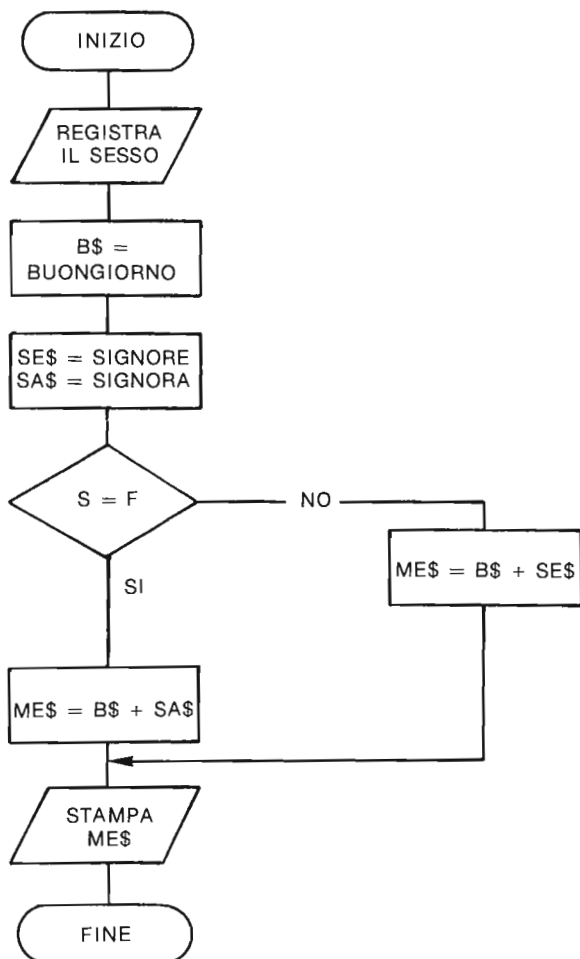
Avremmo potuto ugualmente definire:

```
10      (LET) NO$ = NO$ + CO$
```

In questo caso la variabile NO$ conterrà, alla fine dell'esecuzione, il concatenamento delle stringhe che contengono i nomi e i cognomi.

Trattamento della stringa di caratteri

Al fine di chiarire un po' l'utilità di quanto detto, studieremo un programma che permetterà di stampare "BUONGIORNO SIGNORA" o "BUONGIORNO SIGNORE" a seconda che si risponda M o F ad una domanda relativa al sesso della persona in questione.

Lo schema a blocchi è estremamente semplice:

da cui si ottiene il seguente programma

```
10      ENTRA S$
20      B$ = "BUONGIORNO"
30      SA$ = "SIGNORA"
40      SE$ = "SIGNORE"
50      SE S$ = "F" ALLORA ME$ = B$ + " " + SA$: VAI A 70
60      ME$ = B$ + " " + SE$
70      STAMPA ME$
80      FINE
```

cioè in BASIC:

```
10      INPUT S$
20      B$ = "BUONGIORNO"
30      SA$ = "SIGNORA"
40      SE$ = "SIGNORE"
50      IF S$ = "F" THEN ME$ = B$ + " " + SA$: GO TO 70
60      ME$ = B$ + " " + SE$
70      PRINT ME$
80      END
```

Così, al momento dell'esecuzione, se si batte

?F ®   o   ?M ®

si ottiene:

□ BUONGIORNO SIGNORA   □ BUONGIORNO SIGNORE

Nota. — Nelle istruzioni 50 e 60 si è aggiunto il carattere bianco rappresentato da « », per ottenere una separazione tra le due parole.

Le funzioni di trattamento sulle stringhe di caratteri

Allo stesso modo in cui esistono funzioni matematiche, esistono un certo numero di funzioni relative alle stringhe di caratteri.

1. LA FUNZIONE LEN

Questa funzione, che, in italiano, può essere chiamata LUN, permette di ottenere la lunghezza, in numero di caratteri, della stringa.

Esempio:

Se si scrive

- PRINT LEN ("PERA") ®, si ottiene
  □ 4
- PRINT LEN ("MICROELABORATORE") ®, si ottiene
  □ 16

2. LE FUNZIONI ESTRAZIONE DI CARATTERI

L'operazione concatenamento permette di aggiungere dei caratteri ad una stringa. È dunque necessario disporre di funzioni che permettano di estrarre una sotto-catena di caratteri, partendo da una catena esistente. Per fare ciò, nel BASIC esteso, sono disponibili tre funzioni:

RIGHT$, LEFT$ e MID$.

a) *La funzione di estrazione RIGHT$ (DESTRA)*

Funziona con due parametri:

RIGHT$ (X$, K)

Il primo parametro è una stringa di caratteri (costante o variabile).
Il secondo è un numero intero (costante o variabile).
Questa funzione permette di estrarre i K caratteri più a destra nella stringa X$.

Esempio:

– PRINT RIGHT$ ("GIANPAOLO", 5) ®
□ PAOLO

b) *La funzione di estrazione LEFT$ (SINISTRA)*

Questa funzione ha le stesse regole della precedente:

LEFT$ (X$, K).

Il primo parametro è una catena di caratteri, il secondo è un numero intero.
Permette di estrarre dalla stringa X$ i K caratteri più a sinistra.

Esempio:

– PRINT LEFT$ ("GIANPAOLO", 4) ®
□ GIAN

c) *La funzione di estrazione MID$ (TRA)*

Questa funzione permette di estrarre una stringa di caratteri all'interno di un'altra stringa. Si serve di due parametri:

MID$ (X$, K, L).

X$ è una stringa di caratteri (costante o variabile).

K è un intero che indica a partire da quale carattere si deve incominciare l'estrazione.

L indica il numero di caratteri da estrarre.

Esempio:

– PRINT MID$ ("L'ESEMPIO TI CHIARISCE", II, 2)
  ☐ TI

In questo esempio, si sono estratti due caratteri a partire dall'undicesimo. Si vede quindi che la funzione MID$ è la più ampia, in quanto sostituisce anche le funzioni RIGHT$ e LEFT$.

Un semplice esempio di programma che lavora su testi

Si tratta di un programma che permette di scrivere una parola in senso inverso (da destra verso sinistra).

```
10      ENTRA A$
20      K = LUN (A $)
30      B$ = " "
40      PER I = 1 A K
50      A$ = SINISTRA$ (A$, K – I + 1)
60      L$ = DESTRA$ (A$, 1)
70      B$ = B$ + L$
80      PRO$ I
90      STAMPA B$
```

Spiegazione. – AS contiene la parola o la frase da scrivere partendo dall'ultima lettera, K contiene la lunghezza della catena, BS incomincia con una stringa vuota.

Nell'anello PER, che prosegue fino all'80, si riduce di 1 la catena iniziale per ogni iterazione e si recupera l'ultimo carattere a destra per ricostituire la parola inversa dentro B$.

Con notazioni BASIC si ha:

```
10      INPUT A$
20      K = LEN (A$)
30      B$ = " "
40      FOR I = 1 TO K
50      A$ = LEFT$ (A$, K – I + 1)
60      RIGHT$ (A$, 1)
```

```
 70     B$ = B$ + L$
 80     NEXT I
 90     PRINT B$
100     END
```

Eseguendo il programma si avrà:

```
?       BANANE ®
□       ENANAB
```

Esercizi sulle operazioni sulle stringhe

1. *Scrivere un programma BASIC che permetta di verificare la presenza di un carattere in una stringa.*
2. *Scrivere un programma che calcoli la frequenza di un carattere in una stringa.*
3. *Scrivere un programma che, a partire dall'esempio dato nel paragrafo precedente, determini se una parola o una frase è palindroma, cioè se una stringa di caratteri è identica nei due sensi di lettura.*

RISOLUZIONE DEGLI ESERCIZI

1.

```
 10   INPUT A$, L$
 20   LG = LEN (A$)
 30   FOR I = 1 TO LG
 40   C$ = MID$ (A$, I, 1)
 50   IF C$ = L$ THEN 90
 60   NEXT I
 70   PRINT "IL CARATTERE"; L$; "NON COMPARE"
 80   GO TO 100
 90   PRINT "IL CARATTERE"; L$; "COMPARE"
100   END
```

Altra soluzione

```
 1      M$ = "NON COMPARE"
10÷40   come nel programma precedente
50      IF C$ = L$ THEN M$ = "COMPARE"
60      NEXT I
70      PRINT "IL CARATTERE"; L$; M$; "IN"; A$
80      END
```

2. La soluzione del secondo problema si ricava dalla soluzione 1, contando il numero di caratteri ogni volta che il test C$ = L$ è vero: si ha dunque:

```
10      N = 0
```

```
20   INPUT A$, L$
30   LG = LEN (A$)
40   FOR I = 1 TO LG
50   C$ = MID (A$, I, 1)
60   IF C$ = L$ THEN N = N + I
70   NEXT I
80   PRINT "IL CARATTERE"; L$; "APPARE";
     N; "VOLTE IN"; A$
90   END
```

Esecuzione

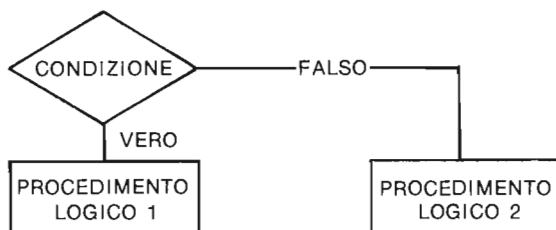
```
?    CHE BEL PROGRAMMA, A
     □ IL CARATTERE A APPARE 2 VOLTE
     IN CHE BEL PROGRAMMA
```

3. È sufficiente per rovesciare una parola riprendere il programma visto nel paragrafo precedente, aggiungendogli le seguenti istruzioni:

```
 15  P$ = A$
 85  IF P$ = A$ THEN 110
 90  PRINT P$; "NON RISULTA PALINDROMO"
100  GO TO 120
110  PRINT P$; "RISULTA PALINDROMO"
120  END
```

4-4. Le istruzioni di test o istruzioni condizionate

Sono istruzioni che danno la possibilità di tradurre degli algoritmi nel linguaggio di programmazione. In pratica, seguendo i valori di certi dati o variabili, deve essere possibile programmare il test di una condizione relativa a queste variabili e di prevedere dei procedimenti logici differenti, a seconda che la condizione sia verificata o no.

La rappresentazione di un test è data dal seguente schema a blocchi:

Lo schema può essere così descritto:
SE la condizione è vera, ALLORA effettua i passaggi logici 1, ALTRIMENTI effettua i passaggi logici 2.

Si vede su questo schema che l'uscita di un test è composta da due rami. Essendo un programma scritto sotto forma di istruzioni in sequenza, questa struttura implica una istruzione del tipo SE... ALLORA... ALTRIMENTI..., con ciascuna delle precedenti parole chiave separate da istruzioni corrispondenti ai due termini dell'alternativa. Questa istruzione non è utilizzata sui BASIC standard, benchè sia stata proposta su alcuni interpretatori. L'altro modo di programmare una tale struttura è di limitarsi ad una istruzione del tipo:

SE ... ALLORA

L'altro termine dell'alternativa (ALTRIMENTI) viene rappresentato nel seguito del programma. In questo caso, la sequenza di istruzioni che segue l'istruzione condizionata rappresenta il procedimento da effettuare quando la condizione non è verificata.

Istruzioni condizionate nel BASIC

Nel BASIC è più usata la seconda soluzione. Si ha dunque una istruzione del tipo:

SE condizione ALLORA istruzione I
istruzione seguente

Le espressioni condizionate

La condizione del test è espressa sotto forma di espressione condizionata o relazionale di cui bisogna definire la struttura.

Definizione. — Una espressione condizionata semplice è una espressione composta da due espressioni dello stesso tipo (aritmetiche o stringhe di caratteri), separate da un operatore relazionale. Una espressione condizionata è VERA o FALSA.

Operatori relazionali

Gli operatori relazionali nel BASIC sono sei

=	uguaglianza
< >	differenza
>	maggiore
<	minore
> =	maggiore o uguale
< =	minore o uguale

Tabella degli operatori relazionali

Esempi:

$- A <> B$	A differente da B
$- C >= 10$	C maggiore o uguale a 10
$- A < B + C$	A minore a B + C

Queste espressioni sono usuali in matematica. Nel BASIC, esse sono ugualmente utilizzabili per le stringhe di caratteri. Il segno = significa identità tra catena.

Gli operatori $<$ e $>$, utilizzati per comparare stringhe di caratteri, indicano che i codici associati sono minori o maggiori. Praticamente, come s'è visto, il fatto che i codici delle lettere siano numeri crescenti, permette di poter comparare delle stringhe di caratteri seguendo l'ordine alfabetico.

Esempio:

L'espressione A\$ $<$ B\$ permette di definire la condizione: la stringa A\$ è classificata alfabeticamente prima della stringa B\$? Allo stesso modo, A\$ = B\$ rappresenta l'identità tra le due stringhe.

Nota. – Si può dunque supporre che algoritmi di scelta sviluppati su valori numerici siano validi anche per valori alfabetici.

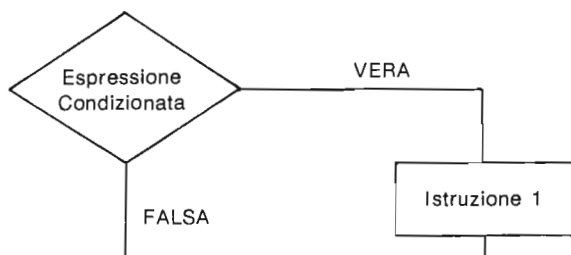
È anche possibile definire espressioni relazionali come:

$$A\$ < B\$ + C\$$$

Attenzione. – Non è invece possibile mischiare i tipi di variabili o costanti usati nella stessa espressione.

Istruzione condizionata semplice

Consideriamo la struttura dell'istruzione BASIC:
SE espressione condizionata ALLORA istruzione I.

Questa struttura può essere così trascritta:

Se l'espressione condizionata è VERA, si esegue l'istruzione 1, altrimenti si passa all'istruzione seguente.

Esempio:

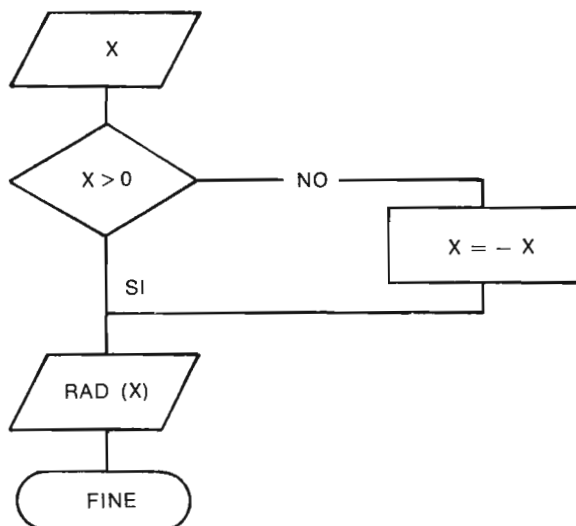
Sia X un numero reale, prima di fare la radice quadra del suo valore assoluto, bisogna assicurarsi che il numero sia positivo, altrimenti dobbiamo prendere l'opposto di X.

Tutto ciò può essere programmato con le seguenti istruzioni:

```
10        INPUT X
20        IF X < 0 THEN X = – X
30        PRINT SQR (X)
40        END
```

Cioè

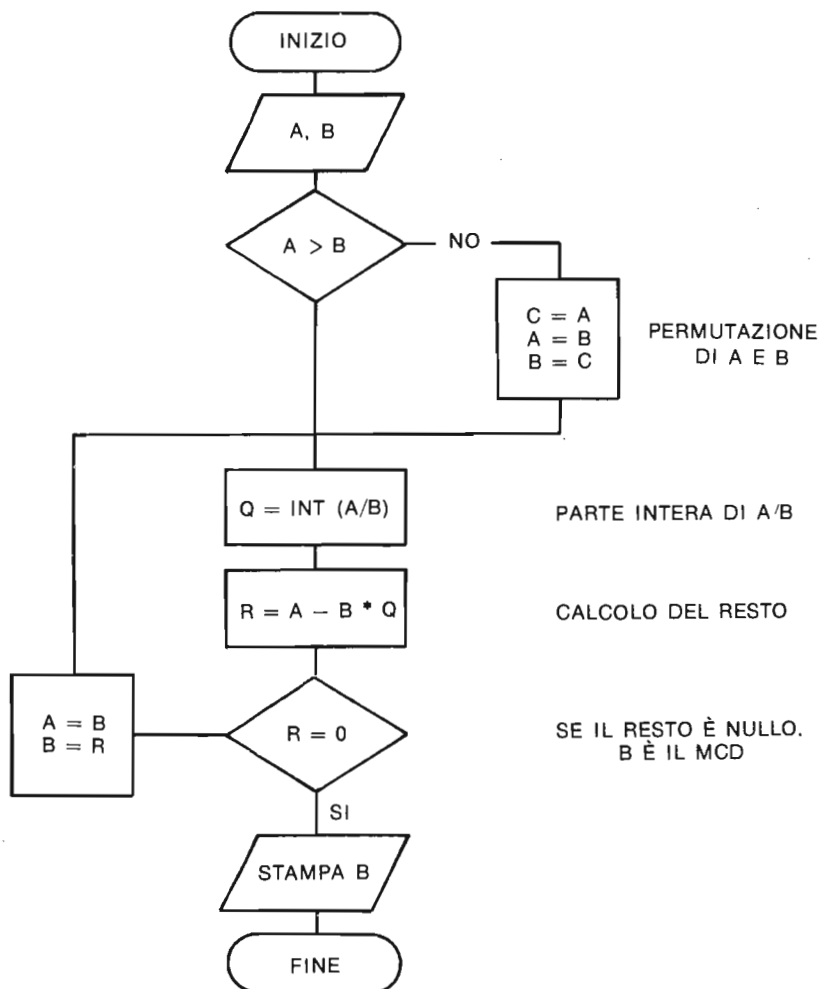
```
10        ENTRA X
20        SE X < 0 ALLORA X = – X
30        STAMPA RAD (X)
40        FINE
```

Lo schema a blocchi corrispondente è:

Se, invece, si ha un problema, dove, a causa del test, la condizione VERO implica un processo logico che necessita di più istruzioni, non si può utilizzare questa istruzione senza far appello ad una diramazione verso un'altra istruzione del programma.

Esempio: Algoritmo di Euclide.

Si abbia lo schema a blocchi dell'algoritmo di Euclide, già definito nel capitolo 2. Si tratta di calcolare il MCD di due numeri A e B.

Nello schema a blocchi si hanno due tests. Il primo serve a determinare il maggiore tra i numeri A e B, poi a scambiarli. Si può, d'altra parte, sottolineare la necessità di utilizzare una terza variabile C per effettuare questa permutazione. Il secondo test serve a sapere se il resto ottenuto è nullo, cosa che indica che B è il MCD.

In ciascuno di questi tests, la condizione negativa non può essere espressa in una sola istruzione, come nell'esempio precedente. Bisogna quindi far ricorso ad una diramazione verso un'altra direzione.

In una espressione condizionata, una tale diramazione si effettua precisando il *numero* dell'istruzione corrispondente dopo la parola chiave THEN (ALLORA). Si può ugualmente farla precedere dalla parola chiave GO TO (VAI A).

Il programma corrispondente è:

```
10    INPUT A, B             ENTRA A, B
20    IF A > B THEN 60       SE A > B ALLORA 60
30    C = A
40    A = B
50    B = C
60    Q = INT (A/B)
70    R = A - B * Q
80    IF R = 0 THEN 120      SE R = 0 ALLORA 120
90    A = B
100   B = R
110   GO TO 60               VAI A 60
120   PRINT B                STAMPA B
130   END
```

PROGRAMMA DELL'ALGORITMO DI EUCLIDE NEL BASIC STANDARD

L'istruzione di salto incondizionato

In questo esempio l'istruzione VAI A (GO TO) è una istruzione di diramazione, che indica il numero di istruzione verso la quale si deve fare il salto. Si sarebbe potuto ugualmente utilizzare questa istruzione dopo il THEN (ALLORA).

Così IF A > B THEN GO TO 60 è equivalente a IF A > B THEN 60.

L'istruzione di salto incondizionato è necessaria nel BASIC, ma non bisogna abusarne, altrimenti potrebbe portare ad un programma «spaghetti», dove si hanno tante diramazioni da non ritrovarsi più.

Non ritorneremo qui sui vantaggi e i meriti della programmazione strutturata, ma, con la maggior parte dei BASIC estesi che dispongono di istruzioni multiple su una linea, è possibile evitare un gran numero di salti.

Riprendendo l'esempio precedente si ottiene:

```
10    INPUT A, B
20    IF A > B THEN C = A : A = B : B = C
30    Q = INT (A/B)
40    R = A - B * Q
50    IF R < > 0 THEN A = B : B = R : GO TO 30
60    PRINT B
70    END
```

PROGRAMMA DELL'ALGORITMO DI EUCLIDE IN BASIC ESTESO

Si può notare che questo programma è leggibile sequenzialmente, che è più compatto, comportando soltanto una diramazione, e che è più veloce rispetto alla versione precedente. Il suo solo inconveniente è di non essere assolutamente standardizzato. Questo programma è stato utilizzato sul BASIC del CBM di Commodore e del sistema APPLE.

Le espressioni condizionate complesse

Sino ad ora abbiamo visto espressioni condizionate semplici che facevano intervenire una sola condizione. In pratica si possono incontrare algoritmi che hanno bisogno di condizioni complesse del tipo:
- Se la condizione 1 *e* la condizione 2 sono vere, allora...
- Se la condizione 1 *o* la condizione 2 sono vere, allora...
- Se la condizione 1 e non la condizione 2 sono verificate, allora...

Tali espressioni sono chiamate espressioni logiche o Booleane (dal nome del matematico Boole che elaborò un'algebra basata su variabili che possono prendere solo 2 valori: vero o falso). Il BASIC esteso permette di programmare direttamente tali espressioni.

Le operazioni logiche

Bisogna innanzitutto definire ciò che si chiama valore e variabile logica (o booleana).

Un valore logico o booleiano è un valore che può assumere solo due stati: il VERO (V) e il FALSO (F), che possono anche essere rappresentati con 0 e 1. Una variabile logica è una variabile che può prendere solo valori logici. Una condizione può dunque essere rappresentata da una variabile logica.

Nel BASIC standard non è possibile definire variabili logiche e utilizzare operazioni logiche.

Ciò che segue è dunque valido solo per i BASIC estesi.

L'operazione negazione logica NON (NOT)

Se si considera una variabile logica A
- NON A è una variabile logica che è vera quando A è falsa e viceversa.

Si può rappresentarle con lo schema seguente detto «tavola della verità».

A	NON A
Vero	Falso
Falso	Vero

Esempio:

Se A rappresenta la condizione: B < C, NON A rappresenterà la condizione inversa B $\geq$ C.

L'operatore E logico (AND)

In questo caso, se si considerano due variabili logiche A e B, l'espressione logica A E B è vera, solo se A e B sono vere *simultaneamente.* Ciò può essere rappresentato con un'altra tavola della verità.

A \ B	Vero	Falso
Vero	Vero	Falso
Falso	Falso	Falso

A E B

Esempio:

– Se A è una condizione C 1 e B è una condizione C 2, la condizione C 1 E C 2 è vera solo se C 1 e C 2 sono vere. In particolare C 1 E C 2 è falsa quando C 1 o C 2 sono falsi.

L'operatore O logico (OR)

L'espressione A O B è vera, se almeno una delle due variabili logiche A o B è vera. Si ha la seguente tavola della verità:

A \ B	Vero	Falso
Vero	Vero	Vero
Falso	Vero	Falso

A O B

Esempio:

– Se A e B rappresentano le condizioni C 1 e C 2, la condizione C 1 O C 2 sarà falsa solo se lo sono simultaneamente sia C 1 che C 2.

Utilizzazione degli operatori logici nel BASIC

Non si può propriamente parlare di variabili logiche nel BASIC. Tuttavia

gli operatori logici sono utilizzabili per definire delle condizioni multiple di una istruzione condizionale.

Esempi:

— Sia da valutare il valore di un numero che deve essere compreso tra un limite inferiore I e uno superiore S.

IF N > I AND N < S THEN...
SE N > I E N < S ALLORA...

— Sia da valutare una stringa di caratteri, per verificare che dia una risposta conforme alla domanda. Si vuole valutare che i caratteri inclusi corrispondono al sesso: (M o F).

```
10    INPUT S$
20    IF S$ = "F" OR S$ = "M" THEN 50
30    PRINT "ERRORE"
40    STOP
50    PRINT "OK"
60    END
```

— Sia da valutare una condizione negativa come l'esclusione di una condizione. Così, nell'esempio precedente, si sarebbe potuto richiedere l'ingresso del dato nel caso che questo fosse errato.

```
10    INPUT S$
20    IF NOT S$ = "F" AND NOT S$ = "M" THEN 10
30    PRINT "OK"
40    END
```

Le istruzioni strutturate in alcuni BASIC

Si è visto che l'istruzione condizionale in BASIC è del tipo:

SE condizione ALLORA istruzione.

Quando la condizione è verificata, si eseguono la o le istruzioni che si trovano dopo l'ALLORA sulla stessa linea, altrimenti, se la condizione non è verificata, si passa all'istruzione in sequenza.

In modo più generale una istruzione condizionale può essere così scritta:

SE condizione ALLORA istruzione 1
ALTRIMENTI istruzione 2

Alcune realizzazioni di BASIC permettono di definire istruzioni di questo tipo. In questo caso, ci si scosta dal BASIC standard. D'altra parte, i gruppi

di istruzioni 1 e 2 sono generalmente inquadrati da strutture che indicano esplicitamente l'inizio e la fine della serie di istruzioni, corrispondenti a ciascun valore della condizione.

Si ha allora una struttura del tipo:

SE condizione ALLORA INIZIA istruzione 1 FINE
ALTRIMENTI INIZIA istruzione 2 FINE

Questo permette in particolare di avere delle strutture condizionali concatenate del tipo:

SE condizione 1 ALLORA INIZIA
SE condizione 2 ALLORA INIZIA
istruzioni FINE
ALTRIMENTI INIZIA
istruzioni FINE
ALTRIMENTI INIZIA
istruzioni FINE

Così il BASIC sviluppato da alcuni costruttori permette strutture del tipo: IF... THEN... ELSE.

La versione più elaborata è presente sui sistemi BASIC sviluppati da ZILOG sui sistemi Z80.

In effetti, vi si trovano strutture del tipo IF

```
IF condizione THEN DO
Istruzioni
DO END
ELSE DO
Istruzioni
DO END
```

Esempio:

L'algoritmo di Euclide, visto precedentemente, diventa:

```
 10    INPUT A, B
 20    IF A > B THEN DO
 30    LET C= A
 40    LET A = B
 50    LET B = C
 60    DO END
 70    REM CALCOLO DEL MCD
 80    LET Q = INT (A/B)
 90    LET R = A - B * Q
100    IF R < > O THEN DO
```

```
110        LET A = B
120        LET B = R
130        GO TO 80
140        DO END
150        ELSE DO PRINT B
160        DO END
170        END
```

Nota. — Questo tipo di struttura è utile specialmente quando si hanno più condizioni annidate l'una nell'altra.

Esercizi e problemi

— *Si abbia il seguente programma BASIC:*

```
10         INPUT X, Y
20         IF X < Y THEN PRINT X "MINORE DI" Y
30         IF X > Y THEN PRINT X "MAGGIORE DI" Y
40         IF X = Y THEN PRINT X "UGUALE A" Y
50         GO TO 10
60         END
```

Questo programma è corretto dal punto di vista sintattico?

Che risultato darà, se entrano i dati X = 50 e Y = 100? Che risultato darà, se X = 20 e Y = 10? In che caso funziona correttamente? Come è necessario modificarlo perchè stampi i risultati?

— *Scrivere un programma che dia una serie di valori non nulli e che calcoli la somma e il valore medio di questi numeri. La fine dei valori è rappresentato dal valore 0.*

— *Scrivere un programma che verifichi se una stringa di caratteri contiene un'altra stringa (o una parola).*

Soluzione degli esercizi

1° Il programma è sintatticamente corretto.
— Per X = 50 e Y = 100 si avrà la stampa di tre messaggi successivi.
— Per X = 20 e Y = 10 si avrà la stampa di due messaggi.
— Il risultato è corretto per X = Y.
— Per correggere questi errori basta aggiungere le istruzioni:

```
25 GO TO 10
35 GO TO 10
```

2° Il problema non pone difficoltà particolari. Diamo dunque direttamente la

soluzione. S contiene la somma, N è il numero di elementi, M è il valor medio:

```
 10    S = 0
 20    N = 0
 30    INPUT X
 40    IF X = 0 THEN 80
 50    S = S + X
 60    N = N + 1
 70    GO TO 30
 80    M = S/N
 90    PRINT "SOMMA ="; S; "VALOR MEDIO ="; M
100    END
```

3° Bisogna aiutarsi con gli esercizi che, nel paragrafo precedente, ricercavano una lettera in una catena.

Qui, utilizzeremo un principio identico: si comparano tutti i caratteri della catena A$ con il primo carattere della catena B$ (rappresentato da L$).

Quando si ha identità, si estrae allora da A$ una sottocatena M$ di lunghezza uguale a B$ e, se si ha identità, si incrementa un contatore. (Si potrebbe ugualmente stampare un messaggio.) Si ottiene:

```
 1     N = 0
10     INPUT A$, B$
15     L$ = LEFT$ (B$, 1)
20     L1 = LEN (A$)
25     L2 = LEN (B$)
30     FOR I = 1 TO L1
40     C$ = MID$ (A$, I, 1)
50     IF C$ < > L$ THEN 60
52     M$ = MID$ (A$, I, L2)
55     IF M$ = B$ THEN N = N + 1
60     NEXT I
70     PRINT "LA PAROLA";B$;"APPARE";N;"VOLTE IN";A$
80     END
```

4-5. L'istruzione di iterazione PER (FOR)

Benchè sia possibile programmare con le istruzioni di assegnazione e di test tutti gli algoritmi immaginabili, si è visto che una delle cose interessanti che una macchina può eseguire è un lavoro ripetitivo o iterativo. Questo lavoro può essere fatto in più modi:

— Se si conoscono il numero di operazioni iterative da eseguire, si procederà azzerando un contatore di iterazioni e, successivamente, alla fine di ogni iterazione, si incrementerà il contatore sino a quando questo sarà arrivato al numero previsto di iterazioni.

— Negli altri casi, non si conosce il numero di iterazioni, ma si conosce il criterio di arresto. Si può dire che si itera FINCHÈ la condizione è verificata o che si RIPETE le iterazioni FINO A CHE una certa condizione è completata (tali strutture sono per esempio disponibili in un linguaggio come il PASCAL).

Nel BASIC è stata prevista solo la prima struttura. Si tratta della struttura di anello di programma PER... PROS (FOR... NEXT). Per ciò che riguarda il secondo tipo di struttura, si può trarsi di impaccio, prevedendo un numero di iterazioni superiore a quello necessario e valutando la condizione che dà il criterio di arresto internamente all'anello. Si può allora uscire prematuramente dall'anello, appena la condizione è verificata.

Struttura dell'istruzione PER in BASIC

Abbiamo già visto alcuni esempi nei capitoli precedenti, tuttavia, la forma generale è più complessa.

PER V = <espressione aritmetica >$_1$ A < espr. aritm. >$_2$
PER PASSO <espr. aritm. >$_3$
Seguito dell'istruzione di iterazione
PROS V
In BASIC si ha:
FOR V = <espr. aritm. >$_1$ TO <espr. aritm. >$_2$ STEP <espr. aritm. >$_3$
Istruzioni
NEXT V.

V è detta variabile di controllo

— L'espressione aritmetica 1, che segue la parola chiave FOR (PER), rappresenta il valore iniziale della variabile di controllo dell'iterazione. In pratica, l'istruzione che segue il FOR può essere una istruzione aritmetica di assegnazione. Può quindi contenere delle funzioni matematiche.

— L'espressione aritmetica 2 rappresenta il valore finale della variabile di controllo.

— La parola chiave STEP (PASSO) precisa il valore che bisogna aggiungere ad ogni operazione. Può quindi essere una espressione aritmetica. Il passo può quindi essere negativo.

Il parametro può essere omesso se è uguale all'unità (1).

Istruzione fine di anello PROS (NEXT)

L'introduzione di una istruzione PER (FOR) presuppone che venga indicato il punto in cui devono terminare le istruzioni su cui si deve lavorare con l'iterazione. Ciò viene fatto grazie all'istruzione PROS (NEXT), seguita dalla variabile di controllo; quest'ultima può essere omessa quando non esistono ambiguità, cioè se nel programma esiste un solo anello.

Esempio:

Riprendiamo qui l'esempio, già dato in un paragrafo precedente, relativo alla somma dei primi N numeri interi. Sia S la somma e N il numero.

```
10      INPUT N
20      S = 0
30      FOR I = 1 TO N
40      S = S + I
50      NEXT I
60      PRINT S
70      END
```

Nota. — Nella maggior parte dei BASIC, se si dimentica l'istruzione di inizio S = 0, il programma funzionerà comunque, in quanto S è azzerata automaticamente.

Un altro esempio permette di stampare i risultati di una funzione con il numero dei punti N, l'incremento X e il valore iniziale dei dati inseriti con la tastiera. Il valore massimo è dunque I + N × X.

Si ottiene:

```
10  INPUT I, N, X              10  ENTRA I, N, X
20  FOR V = I TO I + N * X     20  PER V = I A I + N * X
    STEP X                         PASSO X
30  PRINT V, LOG (V)           30  STAMPA V, LOG (V)
40  NEXT V                     40  PROS V
50  END                        50  FINE
```

Se si esegue questo programma con i dati I = 1, N = 5, X = 1, si ottiene:

```
? 1, 5, 1
1       0
2       .693147181
3       1.09861229
4       1.38629436
5       1.60943791
6       1.79175947
```

Si sarebbe potuto far eseguire il programma con un passo frazionario, ad esempio 0.1.

Struttura iterativa indefinita (tipo 2)

Si prenda l'esempio della ricerca del massimo tra un elenco di numeri battuti sulla tastiera: l'ultimo numero è nullo. In questo caso, si suppone che si faranno iterazioni fino al momento in cui si incontrerà un numero nullo.

```
10 M = 0                        10 M = 0
20 PER I = 1 A 10000            20 FOR I = 1 TO 10000
30 ENTRA X                      30 INPUT X
40 SE X = 0 ALLORA 70           40 IF X = 0 THEN 70
50 SE X > M ALLORA M = X        50 IF X > M THEN M = X
60 PROS I                       60 NEXT I
70 STAMPA "MASSIMO"; M          70 PRINT "MASSIMO = "; M
80 FINE                         80 END
```

Esecuzione di questo programma

```
RUN
?          2 ®
?         23 ®
?         98 ®
?          4 ®
?        456 ®
?          0 ®
□               MASSIMO = 456
```

In questo esempio si ha a che fare con una struttura iterativa del secondo tipo (FINCHÈ o RIPETI FINO A). Qui si è programmato un anello da 1 a 10000, ma, appena si incontra la cifra 0, si esce dall'anello e si stampa il massimo.

Anelli PER annidati

Fino ad ora, abbiamo visto semplicemente alcuni esempi in cui era presente un solo livello di anello. All'interno del primo anello è possibile definirne un secondo, così come si può definirne un terzo all'interno del secondo, e così via. Si possono dunque avere delle strutture del tipo:

```
PER  I1 = . . . . . . . . .
     PER  I2 = . . . . . . . . . .
          PER IK= . . . . . . . . .
          PROS IK = . . . . . . . . .
          . . . . . . . . . . .
     PROS I2
PROS I1
```

Questa struttura è detta ad anelli annidati.

La sola regola da rispettare è che tutto l'anello di livello inferiore sia inserito in un anello di livello superiore; non è possibile quindi avere strutture del tipo:

```
PER  I1 = . . . . . . . .
       PER I2 = . . . . . . . .
PROS I1
       PROS I2
```

Regola 1. — Non si possono avere scavalcamenti tra due anelli PER (FOR). L'anello più interno deve terminare prima o nello stesso tempo dell'anello che lo contiene. Nel caso che i due anelli terminino allo stesso punto, si deve precisare nell'istruzione PROS (NEXT) inizialmente la variabile di controllo più interna, poi la più esterna.

Esempio:

```
10        A$ = "A"
20        B$ = "B"
30        FOR I = 1 TO 5
40        C$ = C$ + A$
50        FOR J = 1 TO 2
60        C$ = C$ + B$
70        PRINT C$
80        NEXT J, I
90        END
```

L'esecuzione di questo programma dà:

```
AB
ABB
ABBAB
ABBABB
ABBABBAB
ABBABBABB
ABBABBABBAB
ABBABBABBABB
ABBABBABBABBAB
ABBABBABBABBABB
```

Si sarebbe ottenuto lo stesso risultato utilizzando:

```
75        NEXT J
80        NEXT I
```

Entrata e uscita da un anello PER (FOR)

Si è visto nell'esempio 3 precedente che l'uscita dall'anello prima del termine del ciclo è sempre possibile. Al contrario, non è possibile entrare in un anello in qualsivoglia punto. Benchè l'istruzione GO TO permetta teoricamente di inserirsi in qualunque punto del programma, è praticamente vietato immettersi in una istruzione contenuta in un anello FOR, partendo da una istruzione che si trova all'esterno di questo anello.

Regola 2. — Non si può inserirsi in un anello PER (FOR), se non dall'istruzione PER (FOR) stessa. Si può, invece, uscire in qualunque punto.

Regole semantiche concernenti i parametri di un anello PER (FOR)

Quando si sono definiti i parametri di un anello, si è visto che ciascuno di essi poteva essere definito con una espressione aritmetica. Ora, se il parametro aritmetico (il PASSO) è *positivo*, ciò implica che il valore dell'espressione iniziale della variabile di controllo sia inferiore o uguale al valore finale.

Al contrario, se il PASSO è negativo, il valore iniziale deve essere superiore al valore finale.

Regola 3. — Siano INIZ e TERM i valori iniziali e finali:
- se PASSO > 0 ALLORA INIZ $\leq$ TERM
- se PASSO < 0 ALLORA INIZ $\geq$ TERM

In pratica, se non si rispetta questa regola, bisogna sapere che, nella maggior parte dei BASIC, l'anello sarà eseguito una sola volta.

Esempio:

Se si scrive il programma seguente e lo si esegue:

```
10        FOR I = I0 TO 1 STEP 1
20        PRINT I
30        NEXT I
40        END
```

Si ottiene

□ 10

Ciò mostra che si è passati una volta nell'anello, per accorgersi che I non può variare. Questa è una annotazione importante: molti errori di programmazione dipendono dal non averla rispettata.

Esercizi

1. *Scrivere un programma che calcoli il Fattoriale N, cioè il prodotto di N numeri interi.*

2. *Scrivere un programma che calcoli i numeri di Fibonacci, rappresentati dalla formula di ricorrenza:*

$$F_{n+2} = F_{n+1} + F_n \text{ con } F_0 = 0, F_1 = 1, n \geq 0$$

L'inizio della sequenza è dunque 0, 1, 1, 2, 3, 5, 8, 13...

3. *Scrivere un programma che permetta di calcolare l'interesse composto mensile di un capitale investito ad un tasso dell'1% annuo.*
4. *Scrivere un programma che permetta di generare la sequenza di lettere seguenti*

   A
   AB
   ABA
   ABAB
   . . . .

Soluzioni

1. Si tratta di un programma simile a quello della somma.

```
10    F = 1
20    INPUT N
30    FOR I = 1 TO N
40    F = F * I
50    NEXT I
60    PRINT F
70    END
```

2.

```
10    F0 = 0
20    F1 = 1
30    INPUT N
40    FOR I = 1 TO N
50    F2 = F1 + F0
55    PRINT F2
60    F0 = F1
70    F1 = F2
80    NEXT I
90    END
```

Esecuzione

```
?   12 ®
     1
     2
     3
     5
     8
    13
```

```
 21
 34
 55
 89
144
233
```

3. Sia C il capitale, I l'interesse annuo e N il numero dei mesi. Si ottiene:

```
10    INPUT C, I, N
20    PRINT "MESE", "CAPITALE", "INTERESSE"
30    FOR J = 1 TO N
40    IN = C * I / 100 / 12
50    PRINT J, C, IN
60    C = C + IN
70    NEXT J
80    END
```

Esecuzione

```
?   1000, 5, 6

    MESE          CAPITALE        INTERESSE
    1             1000            5
    2             1005            5.025
    3             1010.025        5.050125
    4             1015.07513      5.0753756
    5             1020.1505       5.1007525
    6             1025.25125      5.1262562
```

4.

```
 10   INPUT N
 20   C = 0
 30   FOR I = 1 TO N
 40   IF C = 0 THEN C = 1 : C$ = C$ + "A" : GO TO 80
 50   C = 0
 60   C$ = C$ + "B"
 70   PRINT C$
 80   NEXT I
100   END
```

La variabile C prende qui alternativamente il valore 0 e 1 per indicare il carattere da aggiungere.

4-6. Le istruzioni di entrata-uscita

In questo paragrafo, studieremo le possibilità delle istruzioni di entrata-uscita: ENTRA (INPUT), STAMPA (PRINT), LEGGI (READ) e OTTIENI (GET).

L'ISTRUZIONE ENTRA (INPUT)

Questa istruzione, di cui abbiamo visto l'utilizzazione in numerosi esempi, permette di inserire dei dati numerici, o delle stringhe di caratteri, durante l'esecuzione di un programma. La sintassi di questa istruzione è semplicissima:

ENTRA (INPUT) elenco delle variabili.

L'elenco delle variabili può comprendere nomi di variabili numeriche o non numeriche, separate da virgole.

Esempio:

ENTRA N, M, CO$, NO$
(INPUT)

Durante l'esecuzione del programma, questa istruzione provocherà l'emissione di un punto interrogativo: ?.
È in questo momento che bisogna inserire i dati, separandoli da virgole e facendo seguire l'ultimo dato da un Retourn.

Esempio:

? 10, 15, BARBERIS, ANNA ®

Si ha dunque:
N = 10   M = 15   CO$ = "BARBERIS"   NO$ = "ANNA"

Nota. – Quando un dato alfanumerico è inserito con un ordine INPUT, non è necessario utilizzare le virgolette per indicare il contenuto della catena di caratteri. Ma, se si vogliono inserire degli spazi bianchi a sinistra, le virgolette dovranno essere utilizzate. Ciò presuppone dunque, quando si inserisce una serie di dati, che si sappia ciò che il programma attende e in che ordine. Così, praticamente, se il programma è destinato a utilizzatori che non lo conoscono, bisogna indicare chiaramente ciò che si attende prima di domandare un inserimento di dati (vedi qui sotto l'utilizzazione di piccoli messaggi).

Gli errori che possono prodursi nell'esecuzione di un ordine ENTRA (INPUT).

– Se non si inseriscono dati sufficienti e si schiaccia il tasto TORNA (®),

la risposta sarà un doppio punto interrogativo che indica la mancanza dei dati: ??

- Se, al contrario, il numero dei dati è maggiore di quello atteso, non si hanno sempre delle risposte, e i dati sovrabbondanti sono ignorati. Se si hanno dei messaggi, saranno del tipo: ? EXTRA IGNORED (dati superflui ignorati).
- Se si inseriscono dati di tipo non corrispondente a quello atteso, (per esempio stringhe di caratteri al posto di cifre), si avrà in generale un messaggio che indica l'errore e verranno richiesti nuovamente i dati. I messaggi saranno del tipo: "INPUT ERROR TYPE", "ERROR TYPE MISMATCH" ? REDO FROM START (Ritorna all'inizio dei dati d'entrata).

Stampa di un piccolo messaggio prima dell'entrata dei dati.

Con l'aiuto dell'ordine ENTRA (INPUT), è possibile prevedere un piccolo messaggio che precede il punto interrogativo. Per fare ciò, è sufficiente far seguire l'ordine INPUT da una costante stringa di caratteri, seguita da un ; e dalla lista delle variabili.

Si ha dunque una struttura del tipo:

```
10        ENTRA "PICCOLO MESSAGGIO"; elenco delle variabili
          (INPUT)
```

Ciò provocherà, durante l'istruzione, la stampa: PICCOLO MESSAGGIO ?...

Esempio:

Si debbano inserire tre dati numerici, corrispondenti ad una data di nascita. Si potrà scrivere:

```
10        ENTRA "DATA DI NASCITA" G, M, A
          (INPUT)
          DATA DI NASCITA ? 15, 2, 68 ®
```

Tutto ciò permette dunque di precisare per ciascun ordine ENTRA (INPUT) la natura del dato atteso dal programma.

Regole concernenti l'ordine ENTRA:

Si possono riassumere le regole concernenti l'ordine ENTRA (INPUT), con le tre seguenti regole:

Regola 1. – I dati che entrano devono corrispondere all'ordine dell'elenco

delle variabili e al tipo corrispondente a ciascuna variabile. Se si hanno più dati, essi devono essere separati da virgole (,).

Regola 2. – Se si desidera una stampa di un piccolo messaggio questo deve seguire la parola chiave INPUT (ENTRA) e terminare con un ;.

Regola 3. – Se si desidera inserire delle stringhe di caratteri, gli spazi bianchi, a destra o a sinistra, sono soppressi. Se si desidera prenderli in considerazione, bisogna utilizzare delle virgolette. Queste sono ugualmente obbligatorie se la stringa di caratteri contiene una virgola.

L'ORDINE OTTIENI (GET)

Questa istruzione non è disponibile su tutti i BASIC. Questo ordine permette di leggere un solo carattere per volta.

La sintassi dell'ordine GET è simile a quella dell'ordine ENTRA (GET).

GET elenco di variabili

Tuttavia, in pratica, poichè un solo carattere viene letto ogni volta, è necessario avere una sola variabile. Anzi, poichè questa istruzione serve a verificare l'entrata di qualunque carattere dalla tastiera, è preferibile utilizzare una variabile stringa di caratteri. In effetti, l'utilizzazione di una variabile numerica provocherà un errore di esecuzione, se viene inserito un altro valore. D'altra parte, in questo caso, se non viene inserito alcun carattere, il valore corrispondente sarà nullo: ciò permette dunque di differenziare l'inserimento del carattere 0 dall'assenza di caratteri. È importante sottolineare il carattere «*Tempo reale*» dell'istruzione GET.

In effetti, contrariamente all'ordine INPUT, questa istruzione *non attende* affatto che si inserisca un carattere per continuare l'esecuzione del programma. Così, un anello PER può non essere molto chiaro senza un anello che permetta di valutare l'arrivo o l'assenza di un carattere.

Esempio:

Se si desidera inserire un carattere, bisogna inanellare tante volte l'istruzione GET quante sono necessarie per ottenere quel carattere.

```
10      GET X$
20      IF X$ = "" THEN 10
30      PRINT X$
40      END
```

L'istruzione GET è in particolare molto utile per l'inserimento di dati inte-

rattivi di tipo tempo reale, specialmente nei giochi dove lo schermo non può essere annullato da una serie di dati stampati che è tuttavia necessaria per indicare la scelta del giocatore.

L'ISTRUZIONE LEGGI (READ)

Fino ad ora non abbiamo mai parlato di questa istruzione. La ragione è che questa istruzione introduce una confusione nella nozione dei dati. D'altra parte, questa istruzione, che fa parte del linguaggio, è in effetti un anacronismo che dipende dal fatto che dall'introduzione del BASIC poche macchine dispongono di sistemi interattivi o in tempo parziale. Così si potrebbero sviluppare dei programmi ed eseguirli in maniera non interattiva, introducendo le istruzioni READ (LEGGI) e DATA (DATI) associate.

La sintassi dell'istruzione LEGGI è semplicissima:

LEGGI elenco delle variabili
(READ)

La lista delle variabili può essere di qualsiasi tipo (numerica e/o stringhe di caratteri), ciascuna delle variabili essendo separata da una virgola.

Esempio:

LEGGI A, B$, C, D
(READ)

La dichiarazione dei DATI (DATA) associata all'ordine READ (LEGGI)

Una istruzione LEGGI presuppone sempre l'associazione di una istruzione di dichiarazione di dati.

Effettivamente, nel caso di una istruzione READ (LEGGI), i dati sono incorporati sotto forma di dichiarazione DATA (DATI), che dà i valori dei dati associati alle variabili dell'ordine READ. La sintassi della dichiarazione è:
DATA seguito dei valori
(DATI)

Esempio:

```
10        LEGGI A, B, C, D
          (READ)
20        DATA 7, 15.5, 1.3 E – 2, – 54.36.
```

I dati sono ugualmente separati da virgole e i valori sono associati alle variabili corrispondenti definite nell'istruzione LEGGI.

Nell'esempio sopra visto di avrà: A = 7, B = 15.5, C = 1.3 E – 2, D = 54.36.

L'istruzione DATA può anche non seguire l'ordine LEGGI ed essere in qualunque punto del programma. Si possono avere più istruzioni DATA per una stessa istruzione LEGGI (READ), ma, in questo caso, la lettura comincerà dai dati della prima istruzione DATA. Le altre istruzioni DATA saranno considerate come il seguito della prima nell'ordine in cui esse compaiono nel programma. Quando un dato è stato letto, non può più essere riletto (a meno che si utilizzi l'istruzione RILEGGI: cfr. qui sotto). Ciascuna lettura fa dunque avanzare la lettura stessa nella lista dei dati disponibili nel programma.

Nel caso di dati catene di caratteri, come per l'introduzione INPUT, non è necessario mettere le virgolette, a meno che si voglia tener conto degli spazi bianchi all'inizio o alla fine della catena, o a meno che si voglia tener conto della virgola.

Esempio:

```
(LEGGI)
READ A$, B$, C$
DATA TESTO, "ƀPAROLAƀ", "PAROLA 1, PAROLA 2"
(DATI)
```

In questo caso si ha:

```
A$ = TESTO
B$ = ƀ PAROLA ƀ
C$ = PAROLA 1, PAROLA 2
```

Così, se si domanda di leggere più variabili di quante se ne siano date, si avrà un messaggio di errore del tipo "? ERRORE MANCANO DATI" ("? OUT OF DATA ERROR").

Praticamente, durante l'esecuzione, l'interprete mette in evidenza un 'indicatore' che segnala l'indirizzo memoria dei prossimi dati da leggere. Quando la lista è terminata, l'indicatore stampa un simbolo speciale detto «*terminatore*», che indica che non si hanno altri dati.

L'ISTRUZIONE RILEGGI (RESTORE)

Questa istruzione permette di rileggere un elenco di dati riportando l'indicatore all'inizio dello stesso elenco di dati. Per fare ciò, basta utilizzare una istruzione LEGGI (READ), come nel caso della prima lettura.

Esempio:

1	DATI 8, 4, 7, 3, 5, 12, 9, 14, –3, 17	DATA
10	S = 0	
20	PER I = 1 A 10	FOR I = 1 TO 10
30	LEGGI X	READ X
40	S = S + X	S = S + X
50	PROS	NEXT
60	STAMPA S	PRINT S
70	P = 1	P = 1
80	RILEGGI	RESTORE
90	PER I = 1 A 10	FOR I = 1 TO 10
100	LEGGI X	READ X
110	P = P * X	P = P * X
120	PROS	NEXT
130	STAMPA P	PRINT P
140	FINE	END

La prima parte del programma effettua la somma dei 10 numeri dati, la seconda effettua il prodotto degli stessi numeri. L'esempio è stato naturalmente utilizzato soltanto per rilevare l'utilizzo dell'istruzione RILEGGI. Questa avrebbe potuto essere evitata, se si fossero raggruppati i due anelli PER in uno solo, effettuando simultaneamente somma e prodotto.

Precisazioni e precauzioni da prendere per l'uso dell'istruzione LEGGI

Si è visto che l'istruzione LEGGI (READ) non porta niente di nuovo rispetto all'istruzione ENTRA (INPUT); d'altra parte, essa suppone che i dati siano conosciuti al momento della scrittura del programma. Questo tipo di istruzione di entrata è totalmente da evitare, se si desidera far utilizzare un programma da coloro che non conoscono il linguaggio. In questi casi, infatti è facile avere modificazioni non desiderate del programma (errore di battitura, falsa manipolazione ecc.).

L'ordine LEGGI è veramente utile solo nel caso che si tratti di leggere dati stabili o relativamente stabili (cioè poco modificabili, in rapporto al numero di volte in cui il programma è utilizzato). Si tratterà in questo caso di costanti o di parametri fissi in un certo tempo: ad esempio i prezzi unitari di prodotti che non variano tutti i giorni ... ecc. In questo caso, e solo in questo, si avrà interesse ad utilizzare l'istruzione LEGGI.

D'altra parte, si tratta di una cattiva pratica d'uso di una istruzione LEGGI per iniziare una variabile che sarà ulteriormente modificata nel corso del programma. Infatti ciò rischia di produrre errori quando si vuole ritornare all'inizio dell'algoritmo corrispondente.

Si tratta di una istruzione di USCITA dei risultati che, nella maggior parte dei casi sono stampati, da cui il nome dell'istruzione.

Tuttavia, nel caso di un sistema microelaboratore, questa «stampa» viene fatta più soventemente su terminali video. Rimangono quindi solo ragioni storiche per l'uso della parola chiave STAMPA (PRINT). Sarebbe più opportuno utilizzare la parola ESCI (OUTPUT).

La sintassi dell'istruzione è ugualmente semplice:

STAMPA elenco di espressioni
(PRINT)

Nell'elenco, le espressioni possono contenere sia delle variabili, delle costanti numeriche e/o delle catene di caratteri, sia delle funzioni matematiche o catene di caratteri. La separazione dei differenti elementi dell'elenco è ottenuta con virgola (,) o con punto e virgola (;), ma si hanno, a seconda del separatore, differenti effetti.

- Se il separatore è una *virgola*, il dato sarà stampato cominciando da indirizzi fissi della linea, per esempio tutti i dieci caratteri.
- Se il separatore è un *punto e virgola* i dati saranno stampati l'uno dietro l'altro. Ciò implica l'introduzione di caratteri bianchi per separare i risultati.

Nota. — Su alcune macchine, al posto della parola chiave PRINT, si può utilizzare il carattere ?.

Esempio:

? 5 + 6 ® è equivalente a PRINT 5 + 6 ® □ 11

Regole concernenti l'istruzione STAMPA (PRINT)

- L'istruzione PRINT provoca la stampa di almeno una linea.
- Un'istruzione PRINT senza un elenco successivo produce una linea bianca.
- Nella maggior parte dei BASIC un dato numerico con meno di otto cifre significative è espresso in forma decimale (± XXX.XXX).

Se il dato contiene più di otto cifre o è inferiore a 10^{-8}, questo è stampato sotto forma fluttuante con mantissa ed esponente: ± XXX.XXX E ± XX.

- Le stringhe di caratteri devono essere sempre tra virgolette.
- Se l'elenco da stampare termina con un punto e virgola, il carattere se-

guente (altro ordine PRINT o ? inviato al momento di un ordine INPUT) è stampato sulla stessa linea.

Esempio:

10	STAMPA "NOME";	PRINT "NOME";
20	ENTRA N$	INPUT N$

provoca la stampa di
□ NOME?

Il ? è praticamente stampato attraverso l'ordine ENTRA. Le due istruzioni sono dunque equivalenti a:
10 ENTRA "NOME"; N$

LA FUNZIONE DI TABULAZIONE (TAB)

Si tratta di una funzione disponibile nella maggior parte dei BASIC. Permette di realizzare delle impaginazioni. Questa funzione non può essere utilizzata se non in una istruzione di STAMPA (PRINT).

La sintassi di questa funzione è semplicissima.

TAB (N)

Il parametro N è una variabile o una costante intera che specifica la posizione in cui si deve incominciare a stampare.

Esempi:

10 PRINT TAB (4); "MARGINE DI 4"

darà un margine di 4 caratteri bianchi, dopo i quali sarà stampato il messaggio tra virgolette.

```
□        MARGINE DI 4
10       FOR I = 1 TO 10
20       PRINT TAB (I); I; "BIANCHI"
30       NEXT I
```

Esecuzione:

```
1 BIANCHI
 2 BIANCHI
  3 BIANCHI
   4 BIANCHI
    5 BIANCHI
```

6 BIANCHI
  7 BIANCHI
    8 BIANCHI
      9 BIANCHI
        10 BIANCHI

Se la funzione TAB è utilizzata più volte nella stessa istruzione, il parametro N corrisponde alla posizione N dopo l'inizio della linea e non dopo l'ultimo carattere stampato.

Esempi:

– PRINT TAB (4); "*"; TAB (5); "*"
darà come risultato
    □   **

– PRINT TAB (5); "*"; TAB (5); " "
darà come risultato
    □   *

Ciò mostra che non si ha sovrapposizione di due stampe, anche se queste si fanno utilizzando la stessa tabulazione. Ciò è vero anche quando la posizione della tabulazione cade su un carattere già stampato.
– PRINT TAB (5); "TOTO"; TAB (7); "TITI"
darà
    □   TOTOTITI

Allo stesso modo, nella maggior parte dei BASIC, se si utilizzano due tabulazioni in un ordine differente da quanto ci si aspetta, le operazioni vengono fatte come se la seconda tabulazione non esistesse.
– PRINT TAB (10); "TOTO"; TAB (2); "TITI"
darà lo stesso risultato di prima:
    □   TOTOTITI

La funzione TAB può ugualmente essere utilizzata con una espressione aritmetica come parametro. In questo caso, la parte intera del valore assoluto dell'espressione è utilizzata per definire la posizione della tabulazione.

Esempi:

```
– 10    X = 3.14
  20    PRINT TAB (X); X
  30    END
```

dà una tabulazione di 3 (parte intera di X)

```
    □    3.14
–   10   X = 3
    20   PRINT TAB (X↑2); X
```

dà una tabulazione di 9

```
    □    3
–   10   x = 2; Y = 9
    20   PRINT TAB (X↑2 + SQR (Y)); X;Y
```

dà una tabulazione di 7

```
    □    2  9
```

Ciò permette di fare delle tracce *approssimative*, punto per punto, di funzioni, considerando soltanto i loro valori interi positivi.

Esempio:

Traccia di una retta

```
10    INPUT A, B
20    FOR I = 10 TO 1 STEP – 1
30    PRINT TAB (A * I + B); "*"
40    NEXT I
```

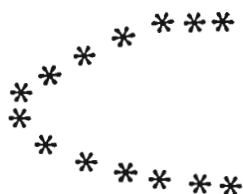
Esecuzione:

? 1, 2 ®   ? 2, 3 ®

pendenza 1   pendenza 2

Altri esempi:

```
–   1    INPUT A
    10   FOR I = 10 TO 1 STEP – 1
    20   PRINT TAB (I ↑ 2/A); "*"
    30   NEXT I
    40   FOR I = 1 TO 10
    50   PRINT TAB (I ↑ 2/A); "*"
    60   NEXT I
```

Permette di ottenere una curva parabolica

```
– 10    FOR I = I TO STEP 0.1
  20    PRINT TAB (ABS (SIN (I))*40); "*"
  30    NEXT I
```

Permette di ottenere la funzione

|SIN (I)|

Esercizi

1. *Scrivere un programma che permetta di inserire degli indirizzi postali sotto la forma:*
   COGNOME? NOME?
   NUMERO E VIA?
   CODICE POSTALE?
   CITTÀ?

   e di scrivere un indirizzo sotto la forma:
   GROSSI GIOVANNA
   Linea bianca
   18 VIA VENETO
   Linea bianca
   20781 BARI

2. *Leggere un elenco di «piccoli messaggi» di prodotti (con un massimo di cinque prodotti e dei relativi prezzi unitari).*
   Inserire un nome di un prodotto, la sua quantità e stampare il prezzo totale.

3. *Scrivere un programma simile al precedente, utilizzando istruzioni GET per inserire un codice dei prodotti e il numero di unità. Si potrà per esempio visualizzare l'elenco dei prodotti:*

   1 PRODOTTO 1 2 PRODOTTO 2
   3 PRODOTTO 3 4 PRODOTTO 4

Dopo l'entrata del codice, si inserisca il codice selezione, il numero di unità e il prezzo sotto la forma:

1 * 5 PREZZO TOTALE = XX.XX

RISOLUZIONI

1.
```
 10  STAMPA "COGNOME NOME"; C$
 20  STAMPA "N° E VIA"; AD$
 30  INPUT "CODICE POSTALE"; C
 40  INPUT "CITTÀ"; L$
 50  PRINT C$
 60  PRINT
 70  PRINT AD$
 80  PRINT
 90  PRINT C; " "; L$
100  END
```
2.
```
 10  READ P1$, P1, P2$, P2, P3$, P3, P4$, P4, P5$, P5
 20  DATA CROISSANT, 2.0 BABÀ, 4 DOLCINO,
     2.5 CROSTATA, 2.3 MERINGA, 3
 30  INPUT "DOLCI"; D$
 40  INPUT "NUMERO"; N
 50  IF DS = P1$ THEN P = P1 * N
 60  IF DS = P2$ THEN P = P2 * N
 70  IF DS = P3$ THEN P = P3 * N
 80  IF DS = P4$ THEN P = P4 * N
 90  IF DS = P5$ THEN P = P5 * N
100  PRINT
110  PRINT "PREZZO TOTALE ="; P
120  END
```

Nota. – Questa maniera di risolvere il problema non è la più concisa, e tiene unicamente conto di ciò che abbiamo visto sinora. Consigliamo al lettore di

riprendere il problema dopo aver visto il capitolo sugli elenchi e le tabelle.

La risoluzione dell'esercizio n° 3 riprende l'esempio del 2, ma supponendo che si inseriscano caratteri puramente numerici che identificano il prodotto e il numero di unità. Supponiamo di disporre di un comando che permette di cancellare tutto lo schermo. (HOME, oppure il carattere speciale ♥).

```
  1     REM LEGGERE I PREZZI UNITARI
 10     READ P1, P2, P3, P4, P5
 20     DATA 1.4,  2.5,  2.8,  3,  2.5,
 25     REM CANCELLARE LO SCHERMO
 30     PRINT "♥" (HOME)
 35     REM VISUALIZZARE I PRODOTTI
 40     PRINT "1 CROISSANT", "2 BABÀ"
 45     PRINT
 50     PRINT "3 DOLCINO", "4 CROSTATA"
 55     PRINT
 60     PRINT "5 MERINGA"
 65     PRINT
 70     REM ENTRA IL CODICE
 80     GET D$
 90     IF D$ = "  "; THEN 80
100     PRINT "  "; D$
110     REM ENTRATA DEL NUMERO
120     GET N
130     IF N = 0 THEN 120
135     REM CALCOLO DEL PREZZO
140     IF D$ = "1" THEN P = P1 * N
150     IF D$ = "2" THEN P = P2 * N
160     IF D$ = "3" THEN P = P3 * N
170     IF D$ = "4" THEN P = P4 * N
180     IF D$ = "5" THEN P = P5 * N
190     PRINT "PREZZO TOTALE="; P
195     REM ATTENDERE IL RITORNO
        PER PASSARE AL SEGUENTE
200     GET R$
210     IF R$ = "  " THEN 200
220     IF ASC (R$) < > 13 THEN 200
230     RESTORE
250     GO TO 10
```

L'esecuzione di questo programma dà sul video:

```
1     CROISSANT        2     BABÀ
3     DOLCINO          4     CROSTATA
5     MERINGA
```

L'ingresso di 2 seguito da 3 dà:

2 * 3 PREZZO TOTALE = 7.5

Se il prodotto 2 (BABÀ), viene richiesto 3 volte, avremo un prezzo totale di 7.5.

Esercizi non risolti

1. *Ci sono molti modi per costruire formule che generano numeri primi. Esistono molti numeri primi che hanno la forma:*
   $n^n + 1 \qquad (1^1 + 1 = 2 \qquad 2^2 + 1 = 5).$
   Scrivere un programma BASIC che determina se $n^n + 1$ è un numero primo per un n variabile da 1 a 7.

2. *La formula $x_n = n^2 + n + 41$ permette di generare numeri primi per n = 1, 2, 3, 4 ...*
   Scrivere un programma che permetta di utilizzare questo generatore di numeri primi per n che va da 1 a 43, e determinare con il programma se il numero ottenuto x_n è primo.

3. *Scrivere un programma che permetta di calcolare il numero 2^{200} e di stamparlo.*

4. *Sia la funzione G_i definita da*
   $$G_i = \frac{\sqrt{5}}{2}\left[\left(\frac{1+\sqrt{5}}{2}\right)^i - \left(\frac{1-\sqrt{5}}{2}\right)^i\right]$$
   Scrivere un programma BASIC che calcoli
   $M_i = INT(G_i = 0.5)$
   con i che varia da 1 a 20; INT definisce la parte intera.

5. *Verificare che la funzione M dia i numeri di Fibonacci:*
   $(F_i = F_{i-1} + F_{i-2}) \qquad F1 = F2 = 1$

CONCLUSIONI AL CAPITOLO 3

In questo capitolo, noi abbiamo affrontato dettagliatamente tutte le istruzioni di base del linguaggio: le istruzioni aritmetiche, di trattamento di catene di caratteri, di test, di iterazione e di entrata-uscita. Consigliamo al principiante di assimilare bene questo capitolo, prima di proseguire.

Nel seguito, affronteremo alcune tecniche più evolute, specialmente il trattamento degli elenchi, delle tabelle, le istruzioni di entrata-uscita a partire da «files» cioè da schedari, le istruzioni di manipolazione del videoterminale...

Daremo esempi più elaborati, è quindi importante aver bene assimilato gli esercizi del Capitolo 3.

CAPITOLO 4

GLI ELENCHI – LE TABELLE LE FUNZIONI I SOTTOPROGRAMMI GLI SCAMBI

1. TRATTAMENTO DEGLI ELENCHI E DELLE TABELLE IN BASIC

In questo capitolo approfondiremo lo studio degli elementi di base del linguaggio, specialmente per quanto concerne il trattamento degli elenchi e delle tabelle. Faremo anche alcuni cenni sulle possibilità di trattamento diretto delle matrici su alcuni BASIC.

Nella seconda parte del capitolo, studieremo le funzioni e i sottoprogrammi. Infine, termineremo con lo studio delle istruzioni di scambio.

1-1. Le nozioni di elenco e di tabella

Fino ad ora gli elementi di base del linguaggio sono state delle variabili semplici che, ad un certo istante, contenevano solo un valore.

Ora, è spesso necessario poter trattare degli insiemi di dati strutturati: ad esempio, una serie di numeri a_i, o una serie di parole di un dizionario p_i..., ecc.

Quando si ha a che fare con problemi matematici, si usano normalmente delle strutture rettangolari o quadrate con righe e colonne ed elementi di tipo a_{ij}.

In matematica, le variabili del tipo a_i, a_{ij} sono dette elementi indici o generici.

NOZIONE DI VARIABILE INDICE

In programmazione, non è possibile definire gli indici con artificio tipografico. È tuttavia possibile definire degli indici associati a nomi delle variabi-

li. In questo caso si parlerà di «variabili con indice». Così a_i è rappresentato da A (I), dove A è un identificatore e I è un identificatore numerico. A (I) è una variabile con indice: A (I) rappresenta un valore generico di una serie di valori numerici: A (1), A (2), A (3), ... A (K) ..... A (N). Qui N è l'indice associato all'ultimo elemento dell'elenco. In programmazione, infatti, non esiste un elenco infinito, da ciò la necessità di dimensionare il numero massimo di elementi di un elenco.

Allo stesso modo, se si vuole rappresentare un elemento a doppio indice, come ad esempio b_{ij}, si definisce una variabile con indice B (I, J), in cui B è un identificatore di variabile e I e J sono variabili numeriche *intere*. In questo caso, B (I, J) rappresenta l'elemento generico di una tabella: B (1, 1), B (1, 2), B (1, 3) .... B (1, N), B (2, 1) .... (B (2, N) .... B (M, 1) .... B (M, N).

In questo esempio, N è l'indice che rappresenta l'ultima colonna della tabella e M è l'indice che rappresenta l'ultima riga. Così gli elementi della tabella possono essere rappresentati da:

B (1, 1) B (1, 2) ...... B (1, N)
B (2, 1) B (2, 2) ...... B (2, N)
....
B (M, 1) B (M, 2).....B (M, N)

Nel caso in cui N = M, si ha una tabella quadrata.

La nozione di tabella permette di rappresentare delle matrici, ma non bisogna confondere le due nozioni, in quanto una tabella può contenere degli elementi che non sono affatto dei coefficienti matriciali. D'altra parte è possibile definire degli elenchi e delle tabelle di stringhe di caratteri. Per fare ciò basta utilizzare un identificatore che termini con il carattere $.

Esempi:

- L$ (K) è una variabile con indici rappresentante l'elemento generico di un elenco di stringhe di caratteri.
  L$ (1) è il primo elemento, L$ (2) il secondo...
- M$ (K, J) è una variabile con indici rappresentante l'elemento generico di una tabella di stringhe di caratteri

Nota. – In certi casi un elenco è chiamato tabella ad una dimensione, in quanto ha un solo indice che varia.

La dichiarazione DIMENSION

Questa dichiarazione è praticamente la sola istruzione di dichiarazione nel linguaggio BASIC (si ha anche la dichiarazione DATA, che abbiamo già visto). Questa permette di definire il numero massimo di elementi di un elenco e

di una tabella. In effetti, l'utilizzazione degli elenchi e delle tabelle presuppone che l'interprete riservi dello spazio in memoria per immagazzinare i differenti elementi degli elenchi o delle tabelle con cui si lavorerà. Ciò viene effettuato con una dichiarazione che deve *precedere* l'utilizzazione di una lista o di una tabella: è la dichiarazione DIM.

La sintassi è semplicissima: basta definire i nomi degli elenchi o delle tabelle, mettendo tra parentesi la o le massime dimensioni dei differenti elementi.

Esempio

— DIM A (10), A$ (20)

Il primo elenco contiene 10 elementi numerici, il secondo 20 elementi stringhe di caratteri.

— DIM T (10, 10), TA$ (5, 10)

La prima tabella contiene 100 elementi numerici organizzati in 10 righe e 10 colonne. La seconda tabella è una tabella di stringa di caratteri organizzata in 10 righe e 5 colonne.

Nota. — Nella maggior parte dei BASIC l'istruzione di dichiarazione DIM è obbligatoria solo se il numero di elementi di un elenco o di una tabella è superiore a 10. Così per valutare dei programmi si può lavorare senza definire alcuna dichiarazione DIM.

Dimensioni variabili

È possibile definire la dimensione con una variabile numerica considerata come un dato che verrà letto al momento dell'esecuzione.

Esempio:

```
10 INPUT        N
20 DIM          B (N)
```

Queste istruzioni permettono di definire un elenco B di dimensioni variabili, essendo N un dato, inserito al momento dell'esecuzione, che precisa la grandezza dell'elenco in quel momento.

Campo di variazione degli indici

Esiste tuttavia una limitazione che riguarda gli indici: devono essere *interi* e *positivi* o *nulli.*

D'altra parte, evidentemente essi devono avere un valore massimo che non superi nè la massima dimensione definita, nè la grandezza disponibile di memoria!

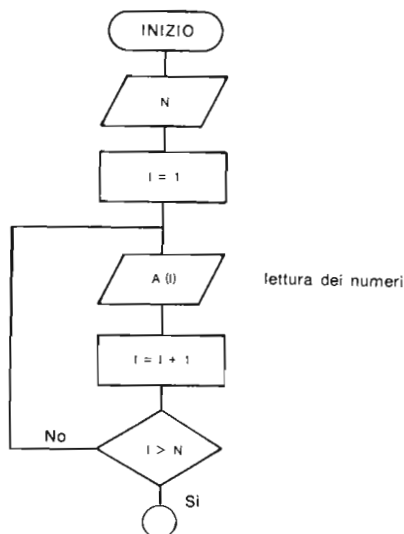
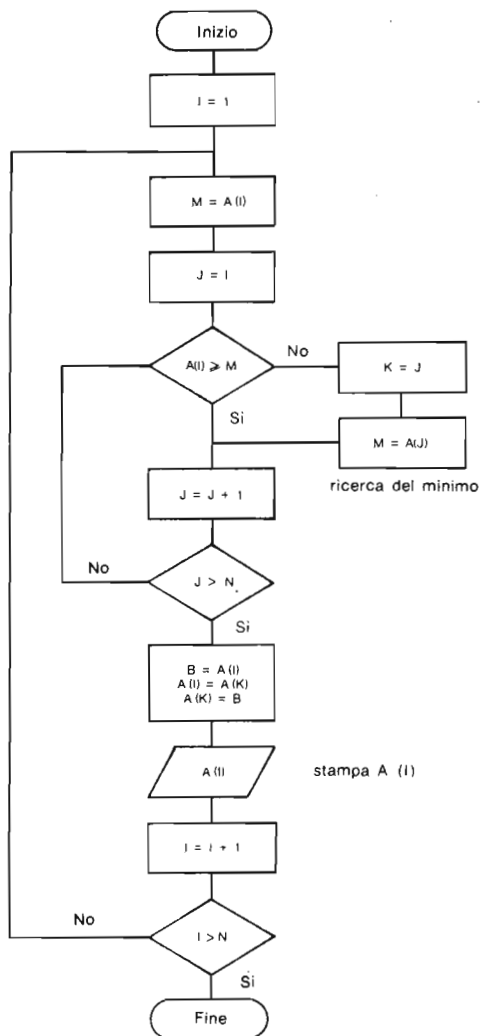
1-2. Il trattamento degli elenchi e delle tabelle

Si è visto che, grazie all'ordine DIMENSION, (facoltativo per i piccoli elenchi o tabelle) è possibile definire delle strutture di elenchi e di tabelle. Non di meno, quando si vogliono trattare tali entità, bisognerà farlo elemento per elemento. Ciò presuppone quindi l'utilizzazione delle istruzioni iterative FOR. In questo paragrafo noi studieremo con alcuni esempi l'uso di tali strutture.

UN PROGRAMMA DI SCELTA

Si debba risolvere il problema della classificazione di una serie di numeri, in modo tale da ordinarli secondo valori crescenti. Esistono più algoritmi di scelta, uno dei più semplici, ma dei meno efficaci in tempo macchina, consiste nel ricercare il minimo e piazzarlo come primo elemento dell'elenco, poi ricercare il prossimo minimo e piazzarlo come secondo, e così via...

Si suppone di inserire in un primo momento il numero di elementi dell'elenco, e successivamente gli elementi stessi.

Lo schema a blocchi del programma è:

Schema a blocchi per il programma di scelta

Il programma corrispondente è:

```
 1      REM ENTRANO I NUMERI N
10      INPUT N
20      DIM A (N)
30      FOR I = 1 TO N
40      INPUT A (I)
50      NEXT I
```

```
55      REM PROGRAMMA DI SCELTA
60      FOR I = 1 TO N
70      M = A (I)
80      FOR J = I TO N
90      IF A (J) < M THEN K = J : M = A (J)
100     NEXT J
110     B = A (I) : A (I) = A (K) : A (K) = B
120     PRINT A (I)
130     NEXT I
140     END
```

In questo esempio abbiamo utilizzato più istruzioni sulla stessa linea: ciò rende il programma più leggibile e compatto.

Utilizzando istruzioni in italiano, si avrà

```
10      ENTRA N
20      DIM A (N)
30      PER I = 1 A N
40      ENTRA A (I)
50      PROS I
60      PER I = 1 A N
70      M = A (I)
80      PER J = 1 A N
90      SE A (J) < M ALLORA K = J : M = A (J)
100     PROS J
110     B = A (I) : A (I) = A (K) : A (K) = B
120     STAMPA A (I)
130     PROS I
140     FINE
```

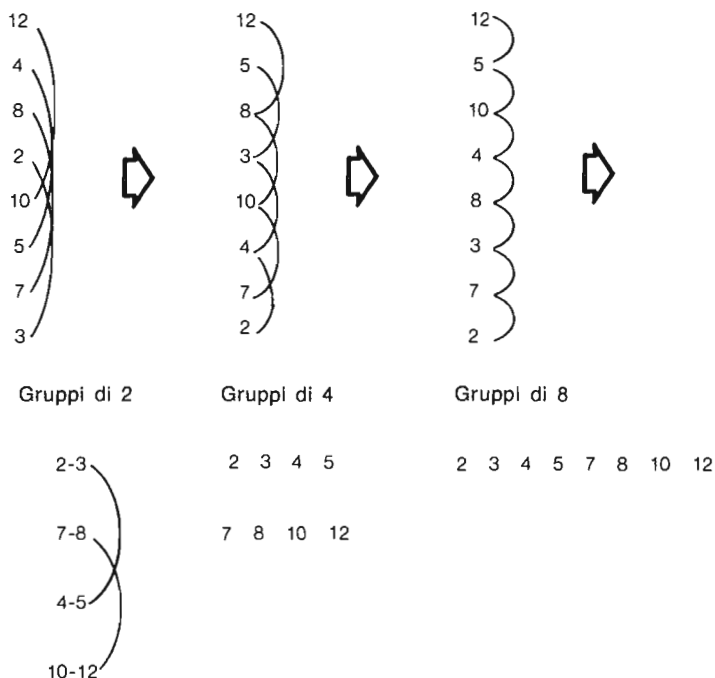
Un altro algoritmo di scelta

L'esempio di algoritmo utilizzato precedentemente è estremamente inefficace, in quanto abbisogna di N^2 comparazioni per ordinare un elenco di numeri. Esiste un altro algoritmo dello stesso tipo, chiamato «tri a bulles», che sarà proposto negli esercizi.

Vediamo qui un altro algoritmo, detto di Shell.

Il principio è il seguente: in un primo momento si comparano gli elementi a due a due: B (I) con B (I + N/2) e li si ordina. Successivamente si fondono i gruppi di due elementi in gruppi di quattro elementi ordinati, poi i gruppi di quattro in gruppi di otto..., fino a quando si arriva al gruppo ordinato completamente.

Esempio

Si abbia la sequenza di numeri:

Il programma corrispondente è dato qui sotto:

```
10    ENTRA N                     INPUT N
20    DIM B (N)
30    PER I = 1 A N               FOR I = 1 TO N
40    ENTRA B (I)                 INPUT B (I)
50    PROS                        NEXT
60    L = N
70    L = INT (L/2)
80    SE L = 0 ALLORA 190         IF L = 0 THEN 190
90    PER J = 1 A N - L           FOR J = 1 TO N - L
100   I = J
110   SE B (I) > = B (I + L)      IF B (I) > = B (I + L)
      ALLORA 170                  THEN 170
120   C = B (I)
130   B (I) = B (I + L)
140   B (I + L) = C
150   I = I - L
```

```
160     SE I > = 1 ALLORA 110     IF I > = 1 THEN 110
170     PROS J                    NEXT J
180     VAI A 70                  GO TO 70
190     PER I = 1 A N             FOR I = 1 TO N
200     STAMPA B (I);             PRINT B (I);
210     PROS                      NEXT
220     FINE                      END
```

Nota. – L'anello J permette di riunire due elenchi, comparando gli elementi B (I) e B (I+L). Esiste un anello più esterno per calcolare L (parte intera di N/2, N/4... fino a 0). Questo algoritmo è più rapido del precedente quando gli elementi sono disposti in maniera aleatoria.

Utilizzazione degli elenchi stringhe di caratteri

Studiamo ora il problema della ricerca di una parola in un dizionario e della sua traduzione. È ovvio che si suppone che il dizionario sia limitato ad una serie di parole ordinate e alla loro traduzione. Si abbia, per esempio, una serie di parole inglesi in un elenco A$, e la loro traduzione in italiano in un elenco B$. Quando si inserisce una parola Q$, la si ricerca nel dizionario: se esiste, viene inserita la traduzione in Q$ e la si stampa, altrimenti si stampa il messaggio "PAROLA SCONOSCIUTA". L'algoritmo è quindi semplicissimo e non è necessario darne altri dettagli.

```
  1     N = 10
 10     DIM A$ (10), F$ (10)
 20     FOR I = 1 TO N
 30     READ A$ (I), F$ (I)
 40     NEXT I
 50     DATA APPLE, MELA, THEN, ALLORA, COMMA,
        VIRGOLA, DATA, DATI, FOR, PER
 60     DATA IF, SE, LET, SIA, PRINT, STAMPA, READ,
        LEGGI, WORD, PAROLA
 70     INPUT Q$
 80     R$ = "PAROLA SCONOSCIUTA"
 90     FOR I = 1 TO N
100     IF Q$ = A$ (I) THEN R$ = F$ (I)
110     NEXT I
120     PRINT R$
130     GO TO 70
140     END
```

Applicazione alla traduzione delle istruzioni BASIC

Nel precedente capitolo abbiamo visto come verificare se una parola esiste in una stringa di caratteri. Abbiamo appena visto come tradurre una parola. Basta dunque fondere i due programmi per ottenere un programma traduttore dei termini BASIC.

Si supponga che l'istruzione in italiano sia inserita in una stringa chiamata BF$. Questa stringa ha come lunghezza L1. Esiste un primo anello di verifica delle parole chiave italiane e, per ciascuna parola chiave, un secondo anello di ricerca dell'esistenza di questa parola chiave nella stringa e la sua sostituzione con la parola chiave inglese.

Si ha dunque schematicamente:

```
PER J =  1 A N
......................................
B$ = PAROLA CHIAVE ITALIANA (J)
......................................
          PER I = 1 A L1
.................................................................
          RICERCA DELLA PAROLA CHIAVE ITALIANA (J) NEL-
          LA STRINGA BF$ E SUA SOSTITUZIONE CON LA PA-
          ROLA CHIAVE INGLESE (I)
          ......................................................
          PROS I
PROS J
```

da ciò si ricava il seguente programma

PROGRAMMA CHE TRADUCE I TERMINI BASIC

```
10    DIM A$ (30), F$ (30)
20    N = 21
30    FOR I = 1 TO N
40    READ A$ (I), F$ (I)
50    NEXT
60    DATA INPUT, ENTRA, NEXT, PROS, THEN, ALLORA,
      DATA, DATI, FOR, PER, IF, SE
61    DATA LET, SIA, PRINT, STAMPA, GO, VAI, TO, A, READ,
      LEGGI, LEFT, SINISTRA
62    DATA RIGHT, DESTRA, SQR, RAD, ON, SU, RETURN,
      RITORNA, INT, INT, RESTORE, RILEGGI
63    DATA END, FINE, GOSUB, SOTPROG, LEN, LUN
65    REM INSERIMENTO DELLE ISTRUZIONI
70    INPUT BF$
80    L1 = LEN (BF$)
```

```
90      FOR J = 1 TO N
95      G$ = " ": D$ = " "
100     B$ = F$ (J)
110     L2 = LEN (B$)
120     L$ = LEFT (B$, 1)
130     FOR I = 1 TO L1
140     C$ = MID$ (BF$, I, 1)
150     IF C$ < > L$ THEN 210
160     M$ = MID$ (BF$, I, L2)
170     IF M$ < > B$ THEN 210
175     IF I - 1 = 0 THEN 185
180     G$ = LEFT (BF$, I - 1)
185     IF L1 - I - L2 + 1 = 0 THEN 200
190     D$ = RIGHT$ (BF$, L1 - I - L2 + 1)
200     BF$ = G$ + A$ (J) + D$
205     L3 = LEN (A$ (J))
206     L1 = L1 - L2 + L3
207     G$ = " ": D$ = " "
210     NEXT I
220     NEXT J
230     PRINT BF$
240     GO TO 70
```

In pratica, il programma presenta un leggero difetto, poiché, pur funzionando bene per la maggior parte delle istruzioni, non funziona assolutamente per istruzioni contenenti dei caratteri, o :. In questi casi, l'interprete si arresta e ignora i caratteri successivi alla (,) o ai (:). Per risolvere la questione, bisogna utilizzare l'istruzione GET.

Così, l'istruzione 70 deve essere sostituita dalla sequenza:

```
70      BF$ = " "
71      GET T$
72      IF T$ = " " THEN 71
73      IF ASC (T$) = 13 THEN 80
74      BF$ = BF$ + T$
75      PRINT T$
76      GO TO 71
```

L'istruzione 73 significa che si è battuto il carattere 13 del codice ASCII, cioè il tasto RETURN: Con questa modifica, il programma permette di tradurre tutte le istruzioni dall'italiano in inglese.

Trattamento delle tabelle

Il trattamento è simile a quello degli elenchi, ma bisogna utilizzare due indici e dimensionare la tabella con un valore massimo per il numero delle righe e uno per il numero delle colonne. Come primo esempio, prendiamo in considerazione la stampa della tavola pitagorica.

```
10   DIM A (10, 10)
20   PER I = 1 A 10          FOR I = 1 TO 10
30   PER J = 1 A 10          FOR J = 1 TO 10
40   A (I, J) = I * J
50   STAMPA A (I, J);        PRINT A (I, J);
60   PROS J                  NEXT J
70   STAMPA                  PRINT
80   PROS I                  NEXT I
90   FINE                    END
```

In uscita si ha:

```
 1   2   3   4   5   6   7   8   9   10
 2   4   6   8  10  12  14  16  18   20
.........................................
10  20  30  40  50  60  70  80  90  100
```

Altro esempio: il triangolo di PASCAL

Il triangolo di PASCAL si ottiene considerando gli elementi della prima colonna uguali ad 1: $a_{11} = 1 \ldots a_{i1} = 1$, mentre gli altri elementi sono ottenuti considerando la relazione:

$$a_{ij} = a_{i-1\,j} + a_{i-1\,j-1} \qquad \text{per } i, j > 1 \text{ e } j \leq i$$

In altre parole, un elemento è uguale alla somma di due elementi: il primo è sopra di lui, mentre il secondo è alla sinistra di quest'ultimo. Si suppone che gli elementi non definiti di una linea siano nulli.

Il programma corrispondente è allora:

```
10   DIM A (10, 10)
15   A (1, 1) = 1
16   PRINT A (1, 1)
20   FOR I = 2 TO 10
25   A (I, 1) = 1
26   PRINT A (I, 1);
30   FOR J = 2 TO I
40   A (I, J) = A (I - 1, J) + A (I - 1, J - 1)
50   PRINT A (I, J);
```

```
60        NEXT J
70        PRINT
80        NEXT I
90        END
```

Eseguendolo, si ottiene:

```
1
1    1
1    2    1
1    3    3    1
1    4    6    4    1
1    5    10   10   5    1
1    6    15   20   15   6    1
1    7    21   35   35   21   7    1
1    8    28   56   70   56   28   8    1
1    9    36   84   126  126  84   36   9    1
```

2. IL TRATTAMENTO DELLE MATRICI

Nei BASIC standard non esistono sempre istruzioni di trattamento di matrici. Tuttavia, è possibile considerare le tabelle a due dimensioni come matrici e, quindi, di programmare le operazioni abituali sulle matrici stesse. Supporremo, nel seguito, di avere sempre matrici quadrate.

2-1. Realizzazione di una matrice unità

La matrice unità è caratterizzata dagli elementi della diagonale principale uguali all'unità.

Esempio:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Una matrice di questo tipo è caratterizzata dagli elementi:

$a_{ii} = 1 \ \forall_i$

e $a_{ij} = 0$ per $i \neq j$

Una matrice di questo tipo può essere generata con il seguente programma:

```
 10       DIM A (10, 10)
 20       PER I = 1 A 10                  FOR I = 1 TO 10
```

```
30     PER J = 1 A 10            FOR J = 1 TO 10
40     A (I, J) = 0
50     SE I = J                  IF I = J
       ALLORA A (I, J) = 1       THEN A (I, J) = 1
60     STAMPA A (I, J)           PRINT A (I, J)
70     PROS J                    NEXT J
80     STAMPA                    PRINT
90     PROS I                    NEXT I
100    FINE                      END
```

2-2. Addizione di due matrici

L'addizione di due matrici consiste nel sommare tra di loro gli elementi dello stesso indice.

Esempio:

C (I, J) = A (I, J) + B (I, J)

Si ottiene allora il seguente programma:

```
10     DIM A (3, 3), B (3, 3), C (3, 3)
15     N = 3
16     REM LETTURA DELLE MATRICI
20     FOR I = 1 TO N
30     FOR J = 1 TO N
40     READ A (I, J), B (I, J)
50     NEXT I, J
60     DATA 1, 2, 3, 4, 2, 1, 0, 3, 5
70     DATA 0, 1, 2, 3, 4, 2, 5, 3, 0
75     REM CALCOLO DELLA SOMMA
80     FOR I = 1 TO N
90     FOR J = 1 TO N
100    C (I, J) = A (I, J) + B (I, J)
110    NEXT J, I
115    REM STAMPA DELLE MATRICI
120    FOR I = 1 TO N
130    FOR J = 1 TO N
140    PRINT C (I, J)
150    NEXT J
155    IF I = INT (K/2) + 1 THEN PRINT "="; : GO TO 170
160    PRINT " "
170    FOR J = 1 TO N
180    PRINT A (I, J)
```

```
190    NEXT J
200    IF I = INT (K/2) + 1 THEN PRINT "+"; : GO TO 220
210    PRINT " "
220    FOR J = 1 TO N
230    PRINT B (I, J)
240    NEXT J
250    PRINT
260    NEXT I
270    END
```

Nota. – In questo programma, la parte di calcolo propriamente detta è estremamente breve (istruzioni dall'80 al 110). Il resto delle istruzioni serve alla lettura delle matrici e, soprattutto, per la stampa, linea per linea, con un minimo di impaginazione (istruzioni dal 130 al 260). Il risultato ottenuto è:

```
3  7  3     1  3  2     2  4  1
3  5  3  =  0  5  1  +  3  0  2
7  7  3     3  2  3     4  5  0
```

Moltiplicazione di una matrice per uno scalare

Da una matrice A (N, N) si ottiene una matrice B (N, N) = K × A (N, N), moltiplicando tutti gli elementi di A per uno scalare K. In altre parole:

$$b_{ij} = k \times a_{ij}$$

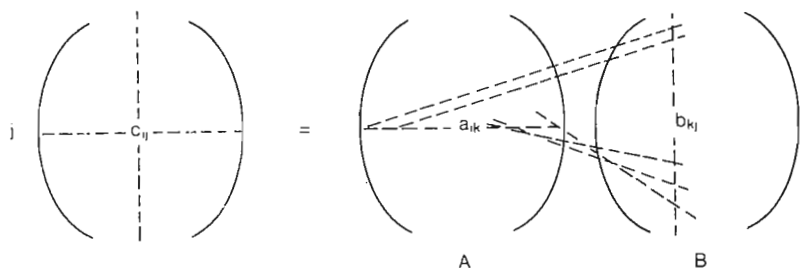
Il programma che permette di effettuare il calcolo è:

```
FOR I =    1 TO N
FOR J =    1 TO N
B (I, J) = K * A (I, J)
NEXT  J, I
```

Applicazioni. – Supponiamo di avere una matrice rappresentante i prezzi unitari (al netto di tasse) di prodotti venduti per quantità. La moltiplicazione per un fattore k uguale alla percentuale di I.V.A. permette di ottenere le tasse da pagare per quei prodotti in ogni singolo caso.

2-3. Moltiplicazione tra due matrici

L'operazione di moltiplicazione è un po' più complessa, in quanto un elemento della matrice prodotto fa intervenire la somma dei prodotti degli elementi della linea I e della colonna J.

$$C_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

In questo esempio, non riprenderemo la parte lettura e stampa delle matrici. Rimane soltanto modificata la parte calcolo (istruzioni dall'80 al 110). Si conservano i due anelli su J e I, ma bisogna aggiungere un terzo anello per permettere di calcolare l'elemento c_{ij} secondo la formula prima vista. Si ha:

```
FOR I =  1 TO N
FOR J =  1 TO N
C (I, J) =  0
FOR K =  1 TO N
C (I, J) =  C (I, J) = A (I, K) x B (K, J)
NEXT K, I, J
```

L'esecuzione, con i precedenti dati, offre:

```
19 14  7     1 3 2     2 4 1
19  5 10  =  0 5 1  x  3 0 2
24 27  7     3 2 3     4 5 0
```

Nota. — È ugualmente possibile moltiplicare due matrici rettangolari, a condizione che il numero delle colonne della prima matrice sia uguale al numero delle righe della seconda. Ad esempio, si può moltiplicare una matrice A (L, N) per una matrice B (N, M), ottenendo una matrice prodotto C (L, M). Per fare ciò è sufficiente modificare il programma precedente, cambiando i termini delle istruzioni FOR:

```
FOR I =  1 TO L
FOR J =  1 TO M
C (I, J) =  0
FOR K =  1 TO N
C (I, J) =  C (I, J) + A (I, K) * B (K, J)
NEXT K, I, J
```

Applicazioni. — Si abbia una matrice dei prezzi di vendita di prodotti fabbricati a differenti prezzi, a seconda delle quantità: si supponga che i prezzi di vendita per quantità siano in colonna e che il prezzo possa essere scomposto in differenti parti (materie prime, fabbricazione, verifica, trasporto). Si abbiano, ad esempio, quattro parti P1, P2, P3, P4, corrispondenti a tre pezzi: T1

per le quantità da 1 a 9, T2 per le quantità da 10 a 99, T3 per le quantità da 100 in su. Si ottiene la matrice B.

		P1	P2	P3	P4	
		5	10	15	20	T1
B	=	4	8	12	15	T2
		3	6	10	10	T3

Si abbiano poi alcune succursali di fabbrica M1, M2, M3, M4, M5, che hanno prodotto ciascuna delle quantità Q1, Q2, Q3 di questi prodotti a questi differenti prezzi: si ottiene allora una matrice A:

		Q1	Q2	Q3	
		10	20	100	M1
		7	30	200	M2
A	=	6	25	150	M3
		5	10	0	M4
		4	11	0	M5

Il problema è di ottenere, per ciascuna fabbrica, il prezzo di vendita corrispondente alle differenti partite P1, P2, P3, P4. Per fare ciò, è sufficiente calcolare per ciascuna partita di ciascuna fabbrica il prezzo di vendita, il cui valore globale sarà:

$$Q1 \times T1 + Q2 \times T2 + Q3 \times T3$$

Esprimendosi in notazione matriciale, tutto ciò significa moltiplicare la matrice A (5, 3) per la matrice B (3, 4), ottenendo la matrice C (5, 4).

		P1	P2	P3	P4	
		430	860	1390	1500	M1
		755	1510	2465	2590	M2
C	=	580	1160	1890	1995	M3
		65	130	195	250	M4
		64	128	192	245	M5

Lasciamo al lettore la cura di scrivere il programma corrispondente e di ottenere il programma che permetta di riempire la matrice C.

SOLUZIONE DELL'ESERCIZIO PRECEDENTE

```
10    DIM B (3,4), A (5,3), C (5,4)
20    FOR I = 1 TO 5
30    FOR J = 1 TO 3
40    READ A (I, J)
50    NEXT I, J
60    DATA 10,20,100,7,30,200,6,25,150,5,10,0,4,11,0
```

```
70      FOR I = 1 TO 3
80      FOR J = 1 TO 4
90      READ B (I, J)
100     DATA 5,10,15,20,4,8,12,15,3,6,10,10
110     NEXT I, J
120     FOR I = 1 TO 5
130     FOR J = 1 TO 4
140     C (I, J) = 0
150     FOR K = 1 TO 3
160     C (I, J) = C (I, J) + A (I, K) * B (K, J)
170     NEXT K
180     PRINT C (I, J)
190     NEXT J
200     PRINT
210     NEXT I
220     END
```

Le istruzioni specifiche di manipolazione delle matrici (MAT)

Negli esempi sopra visti, abbiamo presentato alcuni programmi che permettevano l'effettuazione, con l'aiuto delle istruzioni di base, di calcoli matriciali. Alcuni BASIC dispongono di istruzioni tali da effettuare direttamente queste operazioni, senza dover programmare il calcolo elemento per elemento. Come si vedrà, queste istruzioni non sono necessarie per effettuare il calcolo, ma, quando ci sono, sono certo molto pratiche. Queste istruzioni sono raggruppate nella parola chiave MAT.

Lettura e scrittura dei coefficienti matriciali

È sempre necessario definire una matrice con una dichiazione di dimensioni. Ma, per leggere i coefficienti di una matrice A (N, M), basta scrivere:

MAT READ A (MAT LEGGI A)

Esempio:

Si debba leggere la matrice dell'esempio precedente.

```
10      DIM B (3,4)
20      MAT LEGGI B
30      DATI 5,10,15,20,4,8,12,15,3,6,10,10
```

e cioè

```
10      DIM B (3,4)
20      MAT READ B
30      DATA 5,10,15,20,4,8,12,15,3,6,10,10
```

Alla stessa maniera, è possibile stampare una matrice utilizzando una istruzione

MAT PRINT B; (MAT STAMPA B);

Esempio:

Sempre nell'esempio precedente, volendo stampare B, si avrà:

```
10    DIM B (3,4)
20    MAT LEGGI B                               MAT READ B
30    DATI 5,10,15,20,4,8,12,15,3,6,10,10       DATA ........
40    MAT STAMPA B;                             MAT PRINT B;
50    FINE                                      END
```

Tutto ciò darà in uscita:

```
5   10   15   20
4    8   12   15
3    6   10   10
```

Nota. — In alcuni BASIC è possibile ridimensionare una matrice al momento della sua lettura. In questo caso, le dimensioni della matrice letta sono quelle precisate nell'istruzione MAT LEGGI. Per esempio, si può avere MAT LEGGI A (3, 3), che darà la lettura di una matrice 3 righe e 3 colonne.

Addizione e sottrazione di matrici

Queste istruzioni permettono di precisare l'addizione o la sottrazione di due matrici, utilizzando una sola istruzione di tipo istruzione di assegnazione.

La forma dell'istruzione è:

MAT C = A + B

oppure:

MAT C = A − B

e ciò darà una matrice C somma o differenza tra le matrici A e B.

Addizione

Ciò presuppone che le *dimensioni* delle matrici A e B siano certamente *identiche*. Si possono ugualmente definire le istruzioni di addizione o sottrazione di tipo assegnazione: cioè si può trovare a destra del segno = il nome della matrice risultante che si trova a sinistra del segno =.

Esempio:

MAT A = A + B

oppure:

MAT C = C − A

sono esempi del tutto corretti.

In questo caso però, i contenuti iniziali delle matrici A e C sono distrutti.

Nota. — L'espressione MAT A = B è in genere accettata sui BASIC che dispongono di operazioni MAT. L'espressione permette di specificare l'uguaglianza tra due matrici.

Inizializzazione di matrici particolari

Esistono poi istruzioni che permettono di portare i coefficienti di una matrice al valore nullo o uguale a 1.

a) *Messa a zero di una matrice*

Le istruzioni
MAT A = ZER
o MAT A = ZER (N, M)

permettono di portare i coefficienti della matrice A a zero. Nel primo caso la matrice è già dimensionata, invece, nel secondo la si dimensiona.

Esempio:

MAT A = ZER (2, 2) dà

$$A = \begin{matrix} 0 & 0 \\ 0 & 0 \end{matrix}$$

b) *Messa a uno di una matrice*

Le istruzioni corrispondenti sono:
MAT A = CUN (CON)
MAT A = CUN (N, M)   (CON (N, M))

Nel primo caso tutti i coefficienti della matrice A sono riportati al valore 1. Nel secondo caso la matrice viene, allo stesso tempo, dimensionata.

Esempio:

MAT A = CON (3, 3) dà

$$A = \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

c) *Matrice identità*

Con le istruzioni
MAT A = IDN
o MAT A = IDN (N, N)

si ottiene una matrice che contiene tutti 1 nella diagonale principale e tutti 0 altrove. In ogni caso, la matrice identità deve essere una matrice quadrata a N righe e N colonne.

Esempio:

MAT A = IDN (4, 4) darà:

$$A = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Moltiplicazione di una matrice per uno scalare

La forma generale di tale istruzione è:

MAT A = (espressione aritmetica) * B

Ciò presuppone che le matrici A e B abbiano le stesse dimensioni e che tutti i coefficienti di B siano moltiplicati per un fattore che risulta dal calcolo dell'espressione aritmetica. Questa espressione aritmetica può eventualmente ricondursi ad una variabile numerica o ad una costante numerica.

Esempio:

MAT A = (X + Y) * B

In questo caso, X e Y sono variabili numeriche e A e B sono matrici. Questa istruzione avrà come effetto la moltiplicazione dei coefficienti di A per il risultato del calcolo dell'espressione X + Y.

Moltiplicazione di due matrici

Se si dispone dell'istruzione MAT, si ha la possibilità di programmare la moltiplicazione di due matrici con una sola istruzione. La forma dell'istruzione è:

MAT C = A * B

dove A, B, C sono delle matrici tali che il prodotto di A per B sia possibile, ad esempio, cioè, delle matrici quadrate o rettangolari tali che A (L, M) e B (M, N) diano una matrice C (L, N).

Applicazione all'esempio sopra visto

Il problema studiato per i prodotti venduti a differenti prezzi per differenti quantità può ora essere programmato nella maniera seguente:

```
10        DIM A (5, 3), B (3, 4), C (5, 4)
20        MAT LEGGI A
30        DATI . . . . .
40        MAT LEGGI B
50        DATI . . . . .
60        MAT C = A * B
70        STAMPA "P1", "P2", "P3", "P4"
80        MAT STAMPA C
90        FINE
```

Trasposizione di una matrice

La trasposizione di una matrice significa l'inversione delle righe e delle colonne della matrice. Esiste a tale riguardo una istruzione:

MAT B = TRN (A)

Esempio:

Se:

$$A = \begin{matrix} 4 & 2 & 3 \\ 5 & 6 & 7 \\ 2 & 1 & 4 \end{matrix}$$

dopo averla trasposta, si ha:

$$B = \quad A^T = \begin{matrix} 4 & 5 & 2 \\ 2 & 6 & 1 \\ 3 & 7 & 4 \end{matrix}$$

Se si lavora con matrici rettangolari, la cui dimensione è (N, M), la matrice trasposta ha dimensioni (M, N).

Inversione di una matrice

Quando si considera un prodotto di matrici quadrate $C = A \times B$, se si ottiene come matrice C, la matrice identità, che chiamiamo I, si ha:

$$I = A \times B$$

Si può allora dire che la matrice B è la matrice inversa di A ($B = A^{-1}$). In effetti, come per i numeri razionali, l'inverso di una matrice X è definito con X^{-1}, in modo tale che $X \cdot X^{-1} = 1$.

Tuttavia, nel caso di una matrice, si pone che la matrice inversa non esista.

Bisogna per ciò che questa possieda certe proprietà (il suo determinante deve essere diverso da 0). Non è, comunque, scopo di quest'opera sviluppare nozioni matematiche necessarie ad una formulazione generale e precisa di queste proprietà. Lo scopo di questo paragrafo è di indicare l'esistenza di un'istruzione MAT che permette di calcolare l'inverso di una matrice, quando questo esista.

L'istruzione ha la forma seguente:

MAT B = INV (A)

Applicazioni. — Ci limiteremo ad un semplice esempio di risoluzione di un sistema di equazioni a due variabili.

Si abbia:

$$\begin{aligned} 3x + 4y &= 10 \\ x + 5y &= 7 \end{aligned} \qquad (1)$$

espresso sotto forma matriciale, il sistema può essere scritto:

$$\begin{pmatrix} 3 & 4 \\ 1 & 5 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 10 \\ 7 \end{pmatrix}$$

Si può porre:

$$A = \begin{pmatrix} 3 & 4 \\ 1 & 5 \end{pmatrix}$$

$$X = \begin{pmatrix} x \\ y \end{pmatrix} \quad \text{e} \quad B = \begin{pmatrix} 10 \\ 7 \end{pmatrix}$$

Le matrici X e B sono matrici particolari di dimensione (2, 1), chiamate anche vettori. Il sistema di equazioni (1) può allora scriversi sotto la forma A.X = B.

Per risolvere questo sistema di equazioni senza l'uso delle matrici, si può ad esempio, utilizzare il metodo di sostituzione. Dalla seconda equazione si ricava: x = 7 − 5y, che sostituito nella prima, dà:

$$3\,(7 - 5y) + 4y = 10 \qquad \text{da cui} \qquad 21 - 15y + 4y = 10$$

Da quest'ultima relazione si ha: y = 1, da cui x = 2

Utilizzando la notazione matriciale, si supponga che esista una matrice inversa A^{-1}, per cui

$$X = A^{-1}\,B$$

Per definizione, questa matrice inversa è tale per cui:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \times \begin{pmatrix} 3 & 4 \\ 1 & 5 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

sia: $3\,a_{11} + a_{12} = 1$ $\qquad$ $3\,a_{21} + a_{22} = 0$

$4\,a_{11} + 5\,a_{12} = 0$ $\qquad$ $4\,a_{21} + 5\,a_{22} = 1$

da cui: $a_{11} = 5/11$ $\quad$ e $\quad$ $a_{21} = -1/11$

$a_{12} = -4/11$ $\qquad$ $a_{22} = +3/11$

Si ha dunque

$$A^{-1} = \begin{pmatrix} 5/11 & -4/11 \\ -1/11 & 3/11 \end{pmatrix}$$

Si ha dunque

$$X = A^{-1} \times B$$

Sia

$$X = \begin{pmatrix} 5/11 & -4/11 \\ -1/11 & 3/11 \end{pmatrix} \times \begin{pmatrix} 10 \\ 7 \end{pmatrix}$$

Si ottiene

$$X = \begin{pmatrix} 50/11 - 28/11 \\ -10/11 + 21/11 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

che dà, come soluzione, x = 2, y = 1.

La programmazione di questo problema si riassume allora nelle seguenti istruzioni:

```
DIM     A (2, 2), B (2, 1), X (2, 1), A1 (2, 2)
MAT     READ A
DATA    3, 4, 1, 5
MAT     READ B
DATA    10, 7
MAT     A1 = INV (A)
MAT     X = A1 * B
MAT     PRINT X;
END
```

Nota. — Lo stesso programma può essere utilizzato per risolvere un sistema di equazioni lineari con n incognite, a condizione che abbia una soluzione.

Esercizi

1. *Scrivere un programma di scelta alfabetica.*

2. *Scrivere un programma che permetta di memorizzare uno spartito musicale nella seguente forma. Le note saranno evidenziate così: DO, RE, MI, FA, SOL, LA, SI. L'ottava sarà precisata con il codice 0 seguito da una cifra indicante il numero dell'ottava. I silenzi con la lettera S. I ritmi saranno precisati con un numero che segue la nota: 1 per una nera, 2 per una bianca, 4 per una rotonda, 0.5 per una croma, 0.25 per una biscroma. Le note puntate sono rappresentate da una frazione corrispondente alla durata della nota. Stabilire una tabella corrispondente alla frequenza delle note dello spartito.*

3. *Avendo un intero N e N punti di coordinate (X_i, Y_i) in un sistema di assi cartesiani, scrivere un programma BASIC che determini la distanza tra i punti (X_j, Y_j) e (X_k, Y_k). Si chiede di leggere N, J, K e la tabella dei punti.*

4. *Un quadrato magico è un quadrato diviso in parti, nelle quali i numeri interi, a partire da 1, sono inseriti in maniera tale che le somme effettuate su ciascuna riga, su ciascuna colonna, su ciascuna diagonale siano uguali.*

Esempio:

```
4   9   2
3   5   7   somma = 15
8   1   6
```

Esistono più algoritmi che permettono di ottenere dei quadrati magici di ordine dispari (il numero di elementi per lato è dispari), tuttavia, il più semplice è l'algoritmo che descriveremo ora:

- *la casella proprio sotto il centro è riempita con 1;*
- *i numeri seguenti (2, 3, 4.....) sono inseriti nelle caselle che si trovano all'intersezione della riga inferiore e della colonna di destra;*
- *quando si arriva al bordo del quadrato, si continua all'estremità opposta, seguendo la stessa regola;*
- *se una casella è già stata riempita, si inserisce il numero nella stessa colonna e due righe più sotto.*

Scrivere un programma BASIC che costruisca un quadrato magico, utilizzando l'algoritmo ora descritto.

Esempio:

11	24	7	20	3
4	12	25	8	16
17	5	13	21	9
10	18	1	14	22
23	6	19	2	5

5. *Scrivere un programma che stampi una data, a partire dalla seguente rappresentazione:*

   n, gg, mm, aa

   con n = 1 a 7 per LUNEDÌ, MARTEDÌ, MERCOLEDÌ, GIOVEDÌ, VENERDÌ, SABATO, DOMENICA

   gg = giorno nel mese
   mm = numero del mese
   aa = anno

6. *Modificare il precedente programma per trovare i giorni della settimana corrispondenti tra il 1900 e il 1999. Si terrà conto degli anni bisestili e si supporrà che il primo gennaio 1900 cadeva in martedì.*

Risoluzioni

1. Il principio dell'algoritmo di scelta è simile a quello di scelta per i numeri, ma qui utilizzeremo l'algoritmo di «tri a bulles».

   Si comparano due elementi successivi e, se non sono nel giusto ordine, li si scambia, Per far ciò, è necessario effettuare più «passaggi» e ci si arresta quando non si hanno più scambi nel corso di un passaggio (la variabile E indica se si hanno avuto degli scambi o dei passaggi).

   Il programma è allora:

```
 1  INPUT N
10  DIM A$ (N)
15  REM LETTURA DELLE CATENE DI CARATTERI
20  FOR I = 1 TO N
30  INPUT A$ (I)
40  NEXT I
45  REM INIZIO DI UN PASSAGGIO
50  E = 0
60  FOR I = 1 TO N - 1
70  IF A$ (I) < = A$ (I + 1) THEN 120
80  C$ = A$ (I)
90  A$ (I) = A$ (I + 1)
```

```
100   A$ (I + 1) = C$
110   E = 1
120   NEXT I
125   REM SI CONTINUA FINO A CHE NON SI HANNO
      PIÚ SCAMBI
130   IF E = 1 THEN 50
140   FOR I = 1 TO N
150   PRIN A$ (I)
160   NEXT I
170   END
```

2.

```
 10   REM LETTURA DELLE NOTE
 20   DIM NO$ (12), T (7)
 30   FOR I = 1 TO 12
 40   READ NO$ (I)
 50   DATA DO, RE, MI, FA, SOL, LA, SI, 0, S. «7», «#», «b»
 60   NEXT I
 70   REM LETTURA DEGLI INTERVALLI TRA LE NOTE
 80   FOR I = 1 TO 7
 90   READ T (I)
100   DATA 0, 1, 2, 2.5, 3.5, 4.5, 5.5, 6
110   NEXT I
120   REM INIZIO L = NB DELLE NOTE   OT = OTTAVA
130   L = 100
135   DIM N (100), D (100)
140   OT = 1
150   T2 = 0.5
160   F1 = 65
170   REM LETTURA DELLO SPARTITO
180   FOR I = 1 TO L
190   INPUT N$ (I), D (I)
200   IF N$ (I) = "F" THEN L = I - 1 : GO TO 330
210   G$ = LEFT$ (N$ (I), 2)
220   D$ = RIGHT$ (N$ (I), 1)
230   FOR J = 1 TO 9
240   IF G$ = NO$ (J) THEN 270
250   NEXT J
260   PRINT "ERRORE": GO TO 190
270   IF GS = "O" THEN OT = D (I): GO TO 330
280   IF GS = "S" THEN F (I) = 0: GO TO 330
290   F (I) = 6 * OT + T (J)
300   IF D$ = "#" THEN F (I) = F (I) + T2
310   IF D$ = "b" THEN F (I) = F (I) - T2
```

```
315   F (I) = F1 * 2↑(F (I)/6)
320   NEXT I
330   FOR I = 1 TO L
340   PRINT N$ (I); " " ; D (I); " " ; F (I)
350   NEXT I
360   END
```

4. PROGRAMMA QUADRATO MAGICO

```
  1   REM INSERIMENTO DELLA DIMENSIONE
  5   INPUT N
 10   IF N/2 = INT (N/2) THEN 5
 20   DIM A (N, N)
 30   M = INT (N/2) + 1
 35   REM RIPORTO A 0 GLI ELEMENTI DELLA MATRICE
 40   FOR I = 1 TO N
 50   FOR J = 1 TO N
 60   A (I, J) = 0
 70   NEXT J, I
 80   K = 1
 90   I = M + 1
100   J = M
105   REM CASELLA AL DI SOTTO DEL CENTRO = 1
110   A (I, J) = 1
115   REM CASELLA SEGUENTE
120   K = K + 1
130   IF K > N * N THEN 250
135   REM CASO IN CUI SI ESCE DAL QUADRATO
140   IF I + 1 > N THEN I = 0
150   IF I + 1 < = 0 THEN I = N – 1
160   IF J + 1 > N THEN J = 0
170   IF J + 1 < = 0 THEN J = N – 1
180   REM CASO IN CUI LA CASELLA È GIÀ PIENA
200   IF A (I + 1, J + 1) = 0 THEN 220
210   I = I + 1 : J = J – 1 GO TO 140
220   I = I + 1 : J = J + 1
225   REM RIEMPIRE LA CASELLA
230   A (I, J) = K
240   GO TO 120
245   REM STAMPA DEL QUADRATO
250   FOR I = 1 TO N
260   FOR J = 1 TO N
270   PRINT A (I, J)
280   NEXT J
```

```
290   PRINT
300   NEXT I
310   END
```

Esecuzione

```
   ? 3 ®
             4     9     2
             3     5     7       La somma delle linee
                                 o delle colonne = 15
             8     1     6
   ? 5 ®
            11    24     7    20     3
             4    12    25     8    16
            17     5    13    21     9
            10    18     1     4    22
            23     6    19     2    15
   ? 8 ®    Le cifre pari non sono accettate
   ? 7 ®
            22    47    16    41    10    35     4
             5    23    48    17    42    11    29
            30     6    24    49    18    36    12
            13    31     7    25    43    19    37
            38    14    32     1    26    44    20
            21    39     8    33     2    27    45
            46    15    40     9    34     3    28
```

5.

```
 10   DIM GG$ (7), MM$ (12), NG (12)
 20   FOR I = 1 TO 7
 30   READ GG$ (I)
 40   NEXT I
 50   FOR I = 1 TO 12
 60   READ MM$ (I), NG (I)
 70   NEXT I
 80   DATA  LUNEDÌ, MARTEDÌ, MERCOLEDÌ, GIOVEDÌ,
            VENERDÌ, SABATO, DOMENICA
 90   DATA  GENNAIO, 31, FEBBRAIO, 29, MARZO, 31,
            APRILE, 30, MAGGIO, 31, GIUGNO, 30,
100   DATA  LUGLIO, 31, AGOSTO, 31, SETTEMBRE, 30,
            OTTOBRE, 31, NOVEMBRE, 30, DICEMBRE, 31
110   INPUT N, G, M, A,
120   IF N > 7 OR N < = 0 THEN 110
130   IF M > 12 OR M < = 0 THEN 110
140   IF A > 99 OR A < = 0 THEN 110
```

```
150   IF G < = 0 OR G > NG (M) THEN 110
160   PRINT GG$ (N); G; MM$ (M); A + 1900
170   END
```

3. LE FUNZIONI E I SOTTOPROGRAMMI

Nel capitolo 3 abbiamo visto l'esistenza di un certo numero di funzioni matematiche standard che possono essere utilizzate in espressioni aritmetiche.

Tuttavia, può essere necessario per il programmatore definire nuove funzioni. Queste funzioni saranno utilizzabili in tutto il programma, ma non potranno essere incorporate nel linguaggio vero e proprio. È ciò che studieremo in questo capitolo.

3-1. Studio delle funzioni standard

Prima di passare alla definizione di funzioni particolari, studiamo le funzioni matematiche standard del BASIC.

a) FUNZIONE VALORE ASSOLUTO

È la funzione ABS, avente come parametro o argomento una variabile numerica positiva o negativa.

Y = ABS (X)

permette dunque di ottenere la funzione $Y = | X |$

Esempio:

```
10        ENTRA X
20        Y = ABS (X)
30        STAMPA "VALORE ASSOLUTO DI" ; X ; "=" ; Y
40        FINE
```

Esecuzione

```
?  10
   □  VALORE ASSOLUTO DI 10 = 10
?  −7.14
   □  VALORE ASSOLUTO DI −7.14 = 7.14
```

Questa funzione è particolarmente utile quando si vogliono utilizzare altre funzioni definite unicamente per numeri positivi.

b) LA FUNZIONE ASC

Questa funzione può essere considerata come una funzione aritmetica, nella misura in cui il risultato dà un numero che rappresenta il codice ASCII in decimale (vedi cap. 3) di un qualunque carattere.

Esempio:

```
10      INPUT C$
20      C = ASC (C$)
30      PRINT "IL CODICE ASCII DI"; C$ ; "CORRISPONDE
        A" ; C
40      GO TO 10
50      END
```

Esecuzione

```
?  A ®
   IL CODICE ASCII DI A CORRISPONDE A 65
?  Z ®
   IL CODICE ASCII DI Z CORRISPONDE A 90
```

(Si può notare che 65 + 25 = 90 e si può verificare che i codici delle lettere da A a Z sono le cifre decimali da 65 a 90)

```
?  " " ® rappresenta qui il carattere «bianco»
   IL CODICE ASCII DI    CORRISPONDE A 32
?  0 ®
   IL CODICE ASCII DI 0 CORRISPONDE A 48
?  9 ®
   IL CODICE ASCII DI 9 CORRISPONDE A 57
```

(I codici delle cifre vanno da 48 a 57)

Con questo programma si potranno ottenere i codici di tutti i caratteri A-SCII. Questi codici sono compresi tra 0 e 255.

Nota. — Su alcune macchine si possono avere dei codici non accessibili, poichè la stampante non permette sempre di ottenere tutti i caratteri possibili.

c) LA FUNZIONE RADICE QUADRA RAD (SQR)

Questa funzione è già stata utilizzata in diversi esercizi. Bisogna semplicemente fare attenzione che l'espressione di cui si fa la radice quadra sia positiva (vedi qui sotto). Il termine SQR deriva da «square root»: radice quadra.

Applicazioni: risoluzione di una equazione di secondo grado

Si abbia: $ax^2 + bx + c = 0$

Si sa che, se $a = 0$, ci si riconduce ad una equazione di primo grado. In questo caso, se b e c sono nulli, si ha una indeterminazione; se $c \neq 0$ si ha una possibilità. Nel caso più generale si hanno delle radici reali se il discriminante $b^2 - 4ac$ *è* ≥ 0. In questo caso, le radici sono:

$$x_1, x_2 = (-b \pm \sqrt{b^2 - 4ac}) / 2a$$

```
 10     ENTRA A, B, C
 20     SE A = 0 ALLORA 100
 30     D = B↑2 - 4 * A * C
 40     SE D < 0 ALLORA 80
 50     X1 = (-B + RAD (D))/2 * A
 60     X2 = (-B - RAD (D))/2 * A
 65     STAMPA "X1 ="; X1, "X2 ="; X2
 70     VAI A 150
 80     STAMPA "NESSUNA RADICE REALE"
 90     VAI A 150
100     SE B = 0 ALLORA 130
110     STAMPA "EQUAZIONE DI PRIMO GRADO";
        "X ="; -C/B
120     VAI A 150
130     SE C = 0 ALLORA STAMPA "INDETERMINATA":
        VAI A 150
140     STAMPA "IMPOSSIBILE"
150     FINE
```

Con termini inglesi si avrà

```
 10     INPUT A, B, C
 20     IF A = 0 THEN 100
 30     D = B↑2 - 4 * A * C
 40     IF D < 0 THEN 80
 50     X1 = (-B + SQR (D))/2 * A
 60     X2 = (-B - SQR (D))/2 * A
 65     PRINT "X1 ="; X1; "X2 ="; X2
 70     GO TO 150
 80     PRINT "NESSUNA RADICE REALE"
 90     GO TO 150
100     IF B = 0 THEN 130
110     PRINT "EQUAZIONE DI PRIMO GRADO";
        "X ="; - C/B
120     GO TO 150
```

```
130       IF C = 0 THEN PRINT "INDETERMINATA":
          GO TO 150
140       PRINT "IMPOSSIBILE"
150       END
```

Esecuzione:

```
?    0, 0, 2                          equazione 2 = 0
     IMPOSSIBILE
?    0, 0, 0                          equazione 0 = 0
     INDETERMINATA
?    1, 1, 1                          equazione x² + x + 1 = 0
     NESSUNA RADICE REALE
?    0, 2, 3                          equazione 2x + 3 = 0
     EQUAZIONE DI PRIMO GRADO
     X = -1.5
?    1, -1, -2                        equazione x² - x - 2 = 0
     X1 = 2        X2 = -1
?    1, -6, 9                         equazione x² - 6x + 9 = 0
     X1 = 3.00006104                  X2 = 299993896
```

Nell'ultimo esempio si è visto anche un problema di arrotondamento: avremmo dovuto ottenere una radice doppia $x = 3$. Infatti, se D fosse stato calcolato con l'istruzione seguente

```
D = B * B - 4 * A * C
```

si sarebbe ottenuto X1 = 3 X2 = 3, cosa che mostra come l'operazione di elevamento a potenza possa introdurre errori di arrotondamento.

d) LE FUNZIONI TRIGONOMETRICHE (ATN, COS, SIN, TAN)

Le funzioni trigonometriche standard sono Arcotangente (ATN), Coseno (COS), Seno (SIN), Tangente (TAN). Le si utilizza precisandone un parametro tra parentesi. Il parametro rappresentante la variabile o l'espressione tra parentesi è espresso in radianti per le funzioni COS, SIN e TAN e il risultato di ATN è in *radianti*. Il parametro di queste funzioni può essere espresso sotto forma di espressione aritmetica qualunque, il cui risultato è un numero che esprime un valore in radianti.

Esempi:

```
– PRINT ATN (1) ®
  0.785398163
```

oppure anche:

- PRINT ATN (1) * 4 ®
  3.14159266
  Infatti Arcotangente (1) = π/4
- PRINT COS (0) ®
  1

Alcune tastiere dispongono anche del carattere π, che permette di ottenere direttamente i valori in funzione di π.

- PRINT COS (π/3) ®
  0.5
- PRINT SIN (π/6) ®
  0.5
- PRINT TAN (π/4) ®
  .99999999

In questo caso il risultato teorico è 1, mentre invece si è ottenuto un risultato approssimato per difetto. Ciò è dovuto all'arrotondamento nel calcolo delle funzioni.

Nota. — Se non si dispone del carattere π sulla tastiera, bisogna definire una variabile:

```
10 PI = 3.1459265
```

e utilizzarla successivamente nelle funzioni trigonometriche.

Esempio:

```
20    PRINT SIN (PI/2)
30    END
```

Esecuzione:

□ .99999652

Anche qui il risultato è approssimato, poichè il valore di π è approssimato. Utilizzando l'istruzione PRINT SIN (π/2) ® avremo:

□ 1

Si possono utilizzare multipli di π e ottenere risultati corretti.

Esempio:

PRINT COS (2 * π) ® dà □ 1.

Qualche problema di trigonometria: conversione da radianti in gradi e viceversa

Si sa che $\pi = 180°$

Se il valore di un angolo è dato in gradi (D), il suo valore in radianti (R) è uguale a

$$R = \frac{D}{180} \times \pi$$

Inversamente, se si conosce il valore in radianti, si ha

$$D = \frac{R \times 180}{\pi}$$

Si debbano stampare i valori delle funzioni trigonometriche per angoli dati in gradi: si ha:

```
10    INPUT D
20    R = D * π / 180
25    PRINT "COSENO", "SENO", "TANGENTE"
30    PRINT COS (R); SIN (R); TAN (R)
40    GO TO 10
```

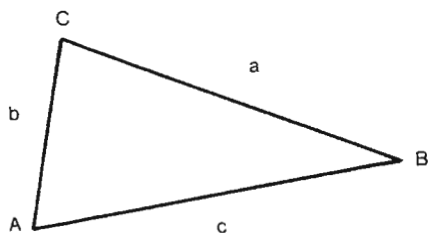
L'esecuzione darà:

```
?   0  ®
□   COSENO        SENO          TANGENTE
    1             0             0
?   30  ®
□   COSENO        SENO          TANGENTE
    0.86602504    0.5           0.577350269
?   45  ®
□   COSENO        SENO          TANGENTE
    0.707106782   0.707106781   0.999999999
?   60  ®
□   COSENO        SENO          TANGENTE
    0.5           0.866025404   1.7320508
?   90  ®
□   COSENO        SENO          TANGENTE
    0             1             ERRORE
                                DIVISIONE PER 0
```

L'ultima operazione dà dunque ERRORE, poichè tang (90°) è uguale a ∞, cosa che qui si traduce in un errore dovuto alla divisione per 0. Poichè cos (90°) = 0, si ha infatti:

$$tgx = \frac{\sin x}{\cos x} = \infty \quad \text{per } x = 90°.$$

Calcolo degli angoli di un triangolo

Si abbia un triangolo i cui lati siano lunghi a, b, c, in corrispondenza ai quali si abbiano gli angoli **A, B, C.**

Applicando il teorema di Pitagora generalizzato, si sa che:

$$a^2 = b^2 + c^2 - 2\,bc \cos A$$

cioè

$$\cos A = \frac{b^2 + c^2 - a^2}{2\,bc}.$$

Non potendo disporre della frazione **ARC** coseno (**A**), si devono utilizzare alcune relazioni trigonometriche, tuttavia non difficili.

$$\sin^2 A + \cos^2 A = 1, \text{cioè} \sin A = \sqrt{1 - \cos^2 A}$$

$$\text{e tg } A = \frac{\sin A}{\cos A} = \frac{\sqrt{1 - \cos^2 A}}{\cos A}$$

da cui:

$$A = \text{Arc tangente} \left(\sqrt{\frac{1 - \cos^2 A}{\cos A}} \right).$$

Alla stessa maniera, partendo da:

$$b^2 = a^2 + c^2 - 2\,ac \cos B$$

si avrà:

$$\cos B = \frac{a^2 + c^2 - b^2}{2\,ac}$$

e:

$$B = \mathit{Arc\ tangente} \left(\frac{\sqrt{1 - \cos^2 B}}{\cos B} \right)$$

C viene invece ottenuto con la relazione

A + B + C = 180° (gradi)

Nota. – Il problema non ha sempre soluzioni, in particolare se si danno ad a, b, c dei valori qualunque.

Effettivamente, se:

$$\frac{b^2 + c^2 - a^2}{2\,bc} > 1$$

non si hanno angoli A reali, cioè in questo caso non si hanno dei triangoli corrispondenti ai lati a, b, c

Il programma corrispondente è dato qui sotto:

```
 10   INPUT A, B, C
 20   CA = (B↑2 + C↑2 – A↑2)/(2 * B * C)
 30   CB = (A↑2 + C↑2 – B↑2)/(2 * A * C)
 40   SA = SQR (1 – CA↑2)
 50   SB = SQR (1 – CB↑2)
 60   AA = ATN (SA/CA) * 180/π
 70   BB = ATN (SB/CB) * 180/π
 80   CC = 180 – AA – BB
 90   PRINT "A ="; A; "B ="; B; "C ="; C
100   PRINT "ANGOLO A ="; A; "ANGOLO B ="; B:
      "ANGOLO C ="; C
110   END
```

L'esecuzione del programma dà:

```
?   2, 3, 4 ®
□   A = 2                 B = 3               C = 4
    ANGOLO A = 28.9550243    ANGOLO B = 46.5674635
    ANGOLO C = 104.477512
?   3, 4, 5 ®
□   A = 3                 B = 4               C = 5
    ANGOLO A = 36.8698977    ANGOLO B = 53.1301023
    ANGOLO C = 90
```

(in effetti, si tratta di un triangolo rettangolo poichè $A^2 + B^2 = C^2$)

```
?   1, 1, 1.414 ®
□   A = 1                 B = 1               C = 1.414
    ANGOLO A = 45.0086517    ANGOLO B = 45.0086519
    ANGOLO C = 89.9826968
```

(si tratta di un triangolo rettangolo isoscele poichè $1.414 \simeq \sqrt{2}$)

```
?   1, 2, 3 ®
□   ERRORE NELL'ISTRUZIONE 40
    (ILLEGAL QUANTITY ERROR IN 40)
```

In questo caso:

$$\frac{B^2 + C^2 - A^2}{2\,BC} = \frac{4 + 9 - 1}{2 \times 2 \times 3} = \frac{12}{12} = 1$$

Se cos A = 1, significa che A = 0 e ciò chiarisce perchè in questo caso non esiste alcun triangolo.

Tuttavia, durante l'istruzione, l'errore non avrebbe dovuto prodursi sull'istruzione 40, poichè:

$$CA = 1 \text{ per cui } SA = \sqrt{1 - CA^2} = 0.$$

In effetti, stampando $1 - CA^2$ prima di effettuare l'istruzione 40, si trova: − 2.25918484 E − 09, vale a dire un valore estremamente vicino a 0, ma negativo! Ciò illustra i problemi di arrotondamento che si possono incontrare in alcuni programmi, in particolare quando i valori ottenuti sono molto vicini ad un valore critico!

e) LE FUNZIONI ESPONENZIALI E LOGARITMICHE

Le funzioni matematiche corrispondenti sono e^x e logaritmo neperiano $\log_e$ (x). Ne abbiamo già visto alcuni esempi. Le forme generali sono EXP (espressione aritmetica) e LOG (espressione).

Esempi:

```
   PRINT EXP (1) ®
□ 2.71828183
   PRINT LOG (1) ®
□ 0
   PRINT LOG (EXP (1)) ®
□ 1
```

Come parametro può essere utilizzata qualunque espressione aritmetica, alla sola condizione che, per il LOG, il valore dell'espressione sia positivo. In particolare possiamo definire le funzioni iperboliche sin h, cos h, tg h, utilizzando la funzione esponenziale:

$$SH = (EXP(X) - EXP(-X))/2$$
$$CH = (EXP(X) + EXP(-X))/2$$
$$TH = \frac{SH}{CH}$$

Vedremo più in là come sia possibile definire queste funzioni una volta per tutte in maniera simbolica in un programma.

Funzione esponenziale con base qualunque

La funzione esponenziale può ugualmente servire a calcolare le funzioni di tipo a^x, utilizzando la relazione:

$$a^x = e^{x \operatorname{Log} a}$$

Ciò può essere anche programmato, utilizzando l'operatore che eleva a potenza (a^x equivale a A↑X)

Esempi:

```
10          INPUT A, X
20          PRINT A↑X, EXP (X * LOG (A))
30          GO TO 10
```

Esecuzione

?	2, 2, ®		
□	4	4	$(2^2 = e^{2 \operatorname{Log} 2})$
?	2, 4.5 ®		
□	22.627417	22.627417	$(2^{4.5} = e^{4.5 \operatorname{Log} 2})$

È chiaro quindi che:
EXP (espressione) è equivalente a e ↑ (espressione)

Funzione logaritmica di base qualunque

La funzione logaritmica neperiana è la funzione inversa dell'esponenziale, poichè, per definizione:

$$\operatorname{Log} e^x = x \text{ e } e^{\operatorname{Log} x} = x$$

In particolare, se si considera un logaritmo in base a, si ha:

$$\operatorname{Log}_a(x) = \operatorname{Log}_a(e^{\operatorname{Log} x})$$
$$\operatorname{Log}_a(x) = \operatorname{Log}_a(e) \times \operatorname{Log}(x)$$

In particolare, se $x = a$, si ottiene:

$$\operatorname{Log}_a(a) = 1 = \operatorname{Log}_a(e) \times \operatorname{Log}(a)$$
$$\operatorname{Log}(a) = \frac{1}{\operatorname{Log}_a(e)}$$

da cui:

$$\operatorname{Log}_a(x) = \frac{\operatorname{Log}(x)}{\operatorname{Log}(a)}$$

Esempio:

```
10      INPUT A, X
20      LA = LOG (X)/LOG (A)
30      PRINT "LOG A BASE"; A; "DI"; X; " ="; LA
40      GO TO 10
```

Esecuzione

```
?       10, 100 ®
        LOG A BASE 10 DI 100 = 2
?       2, 8 ®
        LOG A BASE 2 DI 8 = 3
?       10, 45 ®
        LOG A BASE 10 DI 45 = 1.6532151
?       10, 3 ®
        LOG A BASE 10 DI 3 = 0.477121255
```

f) LE FUNZIONI PARTE INTERA E SEGNO (INT, SGN)

La funzione INT (INT) permette di ottenere la parte intera di un numero decimale o frazionario.

La funzione SGN permette di ottenere il segno di una espressione. Il segno è rappresentato dal valore −1, se il segno stesso è negativo, da +1 se è positivo e da 0 se è nullo.

Esempio:

```
10  ENTRA X                     10  INPUT X
20  STAMPA INT (X); SGN (X)     20  PRINT INT (X); SGN (X)
30  VAI A 10                    30  GO TO 10
```

Esecuzione:

```
?   0 ®
□   0           0
?   −0 ®
□   0           0
?   +0 ®
□   0           0
?   3.5 ®
□   3           1
?   −6.2 ®
```

□ −7 −1
? −0.25 ®
□ −1 −1
? +0.56 ®
□ 0 1

Nota. — La parte intera di un numero negativo corrisponde al numero intero negativo immediatamente inferiore (vedi gli esempi sopra).

Esempio di applicazione

Determinare se un numero intero N è primo. Per fare ciò basta verificare se tutti gli interi a partire da 2 fino alla radice quadrata del numero ($\sqrt{N}$) non sono divisori del numero stesso. Infatti, se non si sono trovati dei numeri per i quali N sia divisibile prima di $\sqrt{N}$, i numeri al di là di $\sqrt{N}$ saranno necessariamente associati ad un fattore che avrebbe dovuto già essere stato trovato come divisore di N, poichè $\sqrt{N} \times \sqrt{N} = N$. Dunque, se si prende un numero $X > \sqrt{N}$, allora l'altro fattore Y deve essere $< \sqrt{N}$.

Per determinare se un numero è primo, è sufficiente verificare se la parte intera della divisione di N per tutti i numeri da 2 a $\sqrt{N}$ è uguale a questa divisione.

Si ottiene allora il seguente programma:

```
10      ENTRA N
20      PER I = 2 A RAD (N)
30      SE N/I = INT (N/I) ALLORA 60
40      PROS I
50      STAMPA N; "RISULTA PRIMO": VAI A 10
60      STAMPA N; "RISULTA DIVISIBILE PER"; I
70      VAI A 10
80      FINE
```

Con termini inglesi avremo:

```
10      INPUT N
20      FOR I = 2 TO SQR (N)
30      IF N/I = INT (N/I) THEN 60
40      NEXT I
50      PRINT N; "RISULTA PRIMO"; GO TO 10
60      PRINT N; "RISULTA DIVISIBILE PER"; I
70      GO TO 10
80      END
```

Esecuzione

```
?  11          ®
   11 RISULTA PRIMO
```

```
?   39              ®
    39 RISULTA DIVISIBILE PER 3
?   37              ®
    37 RISULTA PRIMO
?   1789            ®
    1789 RISULTA PRIMO
?   10789           ®
    10789 RISULTA PRIMO
```

Altri esercizi

Modificare il programma in modo tale da ottenere l'elenco dei numeri primi compresi tra 1 e 1000.

Per ottenere ciò, basta aggiungere un anello più esterno al programma precedente.

```
10    FOR N = 1 TO 1000
20    FOR I = 2 TO SQR (N)
30    IF N/I = INT (N/I) THEN 60
40    NEXT I
50    PRINT N;
60    NEXT N
70    END
```

Esecuzione

```
  1    3    5    7   11   13   17   19   23   29   31
 37   41   43   47   53   59   61   67   71   73
 79   83   89   97  101  107  109  113  ...  ...
```

Esercizi

1. *Si definisce numero perfetto il numero che è uguale alla somma di tutti i suoi divisori, tranne se stesso. Il primo numero perfetto è 6 che è uguale a 1 + 2 + 3, tutti divisori di 6.*
   Scrivere un programma che stampi l'elenco di tutti i numeri perfetti compresi tra 0 e 200.

2. *Si definiscono amici due numeri m e n, quando si verifica che la somma dei divisori dell'uno è uguale all'altro e viceversa.*
   Esempio: 220 e 284 sono amici

Scrivere un programma che permetta di trovare un altro paio di numeri amici.
Nota. — Per trovare il prossimo paio di numeri amici, si deve prevedere di andare fino al 2000.

3. *Tra tutti i numeri interi maggiori di 1, esistono solo quattro numeri che possono essere rappresentati con la somma dei cubi delle loro cifre.*
Ad esempio: $153 = 1^3 + 5^3 + 3^3$
Scrivere un programma che determini gli altri tre
Nota. — I quattro numeri sono compresi tra 150 e 410.

4. *La «prova persa» di FERMAT*
Il matematico FERMAT ha preteso di aver dimostrato un teorema che non è mai stato provato e di cui non si è potuto verificare la non validità. Si tratta di provare che non esistono valori interi a, b, c, tali che $a^n + b^n = c^n$ *per* $n > 2$
1° *Al fine di verificare ciò per un sotto-insieme di numeri interi, si domanda di scrivere un programma che permetta di far variare a, b, c tra 1 e 30 per tutti i valori di n compresi tra 3 e 5.*
2° *Nello stesso tempo si determineranno tutte le terne a, b, c, tali che 1 a b c, per cui:*

$$a^n + b^n - c^n \leq 15$$

RISOLUZIONI

1. *Programma numeri perfetti:*

```
 10   INPUT N
 20   FOR I = 2 TO N
 30   S = I
 40   M = I/2
 50   FOR J = 2 TO M
 60   IF I/J = INT (I/J) THEN S = S + J
 70   NEXT J
 80   IF S = I THEN PRINT I; "RISULTA PERFETTO"
 90   NEXT I
100   END
```

Esecuzione

```
?     30 ®
□      6 RISULTA PERFETTO
      28 RISULTA PERFETTO
```

2. *Numeri amici*

```
 10   INPUT N
 20   FOR I = 2 TO N
 30   S = 1
 40   M = I/2
 50   FOR J = 2 TO M
 60   IF I/J = INT (I/J) THEN S = S + J
 70   NEXT J
 80   IF S = I THEN 180
 90   IF S > I THEN 200
100   M = S/2
110   NS = 1
120   FOR J = 2 TO M
130   IF S/J = INT (S/J) THEN NS = NS + J
140   NEXT J
150   IF NS < > I THEN 200
160   PRINT I; S; "SONO NUMERI AMICI"
170   GO TO 200
180   PRINT I; "RISULTA PERFETTO"
200   NEXT I
210   END
```

Esecuzione

```
?     300 ®
□       6 RISULTA PERFETTO
       28 RISULTA PERFETTO
      284 220 SONO NUMERI AMICI
```

3. *Programma*

```
10   FOR I = 1 TO 9
20   FOR J = 0 TO 9
30   FOR K = 0 TO 9
40   N = I * 10 * 10 + J * 10 + K
50   N3 = I↑3 + J↑3 + K↑3
60   IF INT (N3) = INT (N) THEN PRINT N3; N
70   NEXT K, I, J
80   END
```

Esecuzione

```
153   153
370   370
371   371
407   407
```

Altra soluzione

```
10    FOR I = 1 TO 9
20    FOR J = 0 TO 9
30    FOR K = 0 TO 9
40    N = I * 10 * 10 + J * 10 + K
50    N3 = I * I * I + J * J * J + K * K * K
60    IF N3 = N THEN PRINT N3; "="; I; "↑ 3 +"; J;
      "↑ 3 +"; K; "↑ 3"
70    NEXT K, J, I
80    END
```

Esecuzione

```
153   = 1↑3 + 5↑3 + 3↑3
370   = 3↑3 + 7↑3 + 0↑3
371   = 3↑3 + 7↑3 + 1↑3
407   = 4↑3 + 0↑3 + 7↑3
```

La funzione generatrice di numeri aleatori

Questa funzione, chiamate ALE o, in inglese, RND (Random), permette di dare un numero *a caso* e compreso tra 0 e 1. Normalmente questa funzione deve avere un parametro tra parentesi, ma, in alcuni BASIC, basta precisare RND.

Esempio:

```
1     FOR I = 1 TO 10
2     PRINT RND (1)
3     NEXT I
4     END
```

Esecuzione:

```
0.285986470
0.332769204
0.800480362
0.616685484
0.652093045
0.187796171
0.309893282
0.510722232
0.869524463
0.206369258
```

Nota. — Su alcuni BASIC, rieseguendo lo stesso programma, si ottengono gli stessi numeri; bisognerà allora utilizzare una speciale istruzione (RANDOMIZE), che farà variare la serie delle cifre aleatorie.

- Nella maggior parte dei BASIC realizzati su microelaboratori ciò non è necessario e, dopo ciascuna istruzione, la serie è modificata se si utilizza un parametro negativo RND (—1)....
- Se si avesse utilizzato RND (I) al posto di RND (1), si sarebbe ugualmente ottenuta una serie di numeri aleatori.

Applicazione: Simulazione di un gioco di dadi

Il lancio di un dado è un fenomeno aleatorio che può prendere sei valori: 1, 2, 3, 4, 5, 6. Per ottenere una simulazione di un tal gioco, è sufficiente moltiplicare il risultato della funzione RND per 6 e aggiungere 1 in modo tale da ottenere dei valori che vanno da 1 a 6.9999999..., di cui la parte intera è appunto un numero da 1 a 6.

Nel programma seguente, giocheremo cinque serie di 24 colpi e conteremo quante volte è uscita ciascuna faccia del dado.

```
 10    FOR I = 1 TO 5
 15    REM METTERE A ZERO IL NUMERO DELLE FIGURE
 20    FOR J = 1 TO 6
 30    D (J) = 0
 40    NEXT J
 45    REM LANCIO DEL DADO (24 VOLTE)
 50    FOR K = 1 TO 24
 60    DA = INT (6 * RND (1) + 1)
 70    PRINT DA;
 80    D (DA) = D (DA) + 1
 90    NEXT K
100    PRINT
105    REM STAMPA I RISULTATI
110    FOR J = 1 TO 6
120    PRINT D (J)
130    NEXT J
140    PRINT: PRINT
150    NEXT I
160    END
```

Esecuzione:

```
5 2 2 1 5 5 6 3 5 4 2 3 4
2 4 1 5 5 4 4 5 5 6 1
3 4 2 5 8 2
```

:queste ultime sei cifre rappresentano il numero di uscite per ciascun valore (3 volte 1, 4 volte 2, 2 volte 3, 5 volte 4, 8 volte 5, 2 volte 6)

```
2 5 6 4 4 5 1 6 6 1 3 6 6
3 6 4 6 2 3 2 1 3 1 2
4 4 4 3 2 7

6 2 2 6 1 1 4 6 2 1 4 3 6
2 4 1 4 2 3 3 3 1 5 5
5 5 4 4 2 4

2 1 6 5 3 2 6 6 1 2 1 4 2
4 2 3 1 1 4 6 1 3 2 6
6 6 3 3 1 5

5 6 2 6 5 5 6 5 4 5 2 2 1
2 1 3 6 1 6 5 2 4 2 4
3 6 1 3 6 5
```

Generazione di un numero aleatorio compreso tra due valori A e B (A < B)

L'intervallo di variazione è B – A; essendo il valore A quello iniziale, basta definire

$$N = A + (B - A) * RND(1)$$

Applicazione al gioco del LOTTO

I numeri che possono uscire sono 90 e vengono fatte sei estrazioni. Si ha:

```
10      FOR I = 1 TO 6
20      N = INT (90 * RND (1) + 1)
30      PRINT N;
40      NEXT I
50      END
```

Esecuzione

```
15   56   74   23   55
 4   12   38   69   31
13   35   56   29    4
 2   47   36   78   83
```

Nota. — Questo programma non esclude l'uscita di numeri identici. Se in ogni

elenco non abbiamo avuto l'uscita degli stessi numeri, è solo per nostra fortuna!

Esercizi

1. *Scrivere un programma che giochi a testa o croce. Stampare il risultato sotto forma di un elenco di T o C, contando il numero di T o di C per il numero di colpi che si considera come dato.*

2. *Scrivere un programma che permetta di avere un elenco di numeri a caso, assicurandosi che non si ottenga mai per due volte lo stesso numero.*

3. *Applicare l'esercizio 2 per modificare il gioco del LOTTO.*

Correzione «Testa o croce»

```
 10     INPUT N
 20     T = 0 : P = 0
 30     FOR I = 1 TO N
 40     M = INT (2 * RND (1))
 50     IF M = 1 THEN 80
 60     PRINT "C";
 65     C = C + I
 70     GO TO 100
 80     PRINT "T";
 90     T = T + 1
100     NEXT I
110     PRINT
120     PRINT C; "CROCE"; T; "TESTA";
130     GO TO 10
```

Esecuzione

```
?   10 ®
    TTCCTTCTTC
    6 TESTE   4 CROCI
?   20 ®
    TCCCTTCTCCTCCCCTCCCT
    7 TESTE 13 CROCI
?   10 ®
    TTCCCTCTT
    5 TESTE   5 CROCI
?   20 ®
    CTCTCTTCCCCCCCTCCTTT
    8 TESTE   12 CROCI
```

Si vede in questi esempi che le sequenze variano da una esecuzione all'altra.

Esercizi

1. *Scrivere un programma di conversione decimale binario per numeri aventi al massimo 10 cifre binarie (bits)* (≤ 1023).

2. *Scrivere un programma di conversione di un numero decimale in uno avente base qualunque (ma* ≤ 10).

3. *Scrivere un programma di conversione da decimali in esadecimali (base 16). In questo caso i numeri dal 10 al 15 sono rappresentati dalle lettere A, B, C, D, E, F (cifre esadecimali).*

4. *Scrivere un programma di conversione di un numero a base qualunque (≤ 16) in un numero decimale.*

Correzione degli esercizi

1. *Conversione decimale binaria*

```
10    INPUT N
15    PRINT N; "IN BINARIO ="
20    FOR I = 10 TO 0 STEP - 1
30    B = INT (N/2↑I)
40    N = N - B * 2↑I
50    PRINT B;
60    NEXT I
70    END
```

Esecuzione

```
?    11 ®
□    11 IN BINARIO = 0000001011
?    77 ®
□    77 IN BINARIO = 0000 100 1101
```

2. *Conversione da decimale a base qualunque ≤ 10*

```
 1    INPUT "BASE"; B
 2    IF B > 10 O B < 2 THEN 1
10    INPUT "NUMERO"; N
15    PRINT N; "IN BASE"; B; "=";
20    FOR I = 10 TO 0 STEP - 1
25    D = INT (B↑I)
```

```
30      C = INT (N/D)
40      N = N − C * D
50      PRINT C;
60      NEXT I
70      PRINT
80      GO TO 10
```

Nota. — Il passaggio attraverso la variabile D è necessario per evitare errori d'arrotondamento che, su alcune macchine, possono dare risultati inesatti.

Esecuzione

```
        BASE ? 10 ®
        NUMERO ? 89 ®
□       89 IN BASE 10 = 00000000089
        NUMERO ? 1979 ®
□       1979 IN BASE 10 = 00000001979
        BASE ? 8 ®
        NUMERO ? 45 ®
□       45 IN BASE 8 = 00000000055
        NUMERO ? 1979®
□       1979 IN BASE 8 = 00000003673
        BASE ? 5 ®
        NUMERO 45®
□       45 IN BASE 5 = 00000000140
        BASE ? 3 ®
        NUMERO ? 45 ®
□       45 IN BASE 3 = 00000001200
```

3. *Programma di conversione di un numero decimale in un numero con base qualunque*

```
 1      INPUT "BASE"; B
 2      IF B > 16 OR B < 2 THEN 1
 3      DIM H$ (16)
 4      FOR I = 0 TO 15
 5      READ H$ (I)
 6      NEXT I
 7      DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
10      INPUT "NUMERO"; N
15      PRINT N; "IN BASE"; B; "=";
20      FOR I = 10 TO 0 STEP − 1
25      D = INT (B↑I)
```

```
30    C = INT (N/D)
40    N = N - C * D
50    PRINT H$ (C);
60    NEXT I
65    PRINT
70    GO TO 10
```

Esecuzione:

```
      BASE ? 16 ®
      NUMERO ? 45 ®
□     45 IN BASE 16 = 000000002D
      NUMERO ? 89 ®
□     89 IN BASE 16 = 000000059
```

Nota. — Questo programma è valido per qualunque base inferiore o uguale a 16, per cui può sostituire vantaggiosamente i due programmi precedenti.

4. *Programma di conversione di un numero in base qualunque a un numero in base decimale*

```
 10   INPUT "BASE"; B
 20   DIM A$ (16)
 30   IF B > 16 OR B > 2 THEN 10
 40   FOR I = 0 TO 15
 50   READ A$ (I)
 60   NEXT I
 70   DATA 0,1,2,3,4,5,6,7,7,8,9,A,B,C,D,E,F
 80   INPUT "NUMERO"; N$
 90   PRINT N$; "IN BASE"; B; "="
100   K = LEN (N$)
110   M = 0
120   FOR I = 1 TO K
130   L$ = RIGHT$ (N$, 1)
140   FOR J = 0 TO B - 1
150   IF L$ = A$ (J) THEN 175
160   NEXT J
170   GO TO 80
175   IF K - 1 = 0 THEN 190
180   N$ = LEFT$ (N$, K - 1)
190   M = M + J * B↑(I - 1)
200   NEXT I
210   PRINT M; "IN DECIMALE"
```

```
215   PRINT
220   GO TO 80
230   END
```

Esecuzione:

```
      BASE ? 7®
      NUMERO ? 145®
□     145 IN BASE 7 = 82 IN DECIMALE
      BASE ? 16®
      NUMERO ? 1 AC ®
□     1AC IN BASE 16 = 428 IN DECIMALE
```

3-2. Le funzioni definite da chi sta programmando

Quando si desidera definire qualche funzione che non fa parte della lista standard studiata sinora, è necessario utilizzare una istruzione di *dichiarazione* che comincia con la parola chiave DEF (Definizione). Questa funzione sarà utilizzabile per tutto il resto del programma.

FORMA GENERALE DELLA DICHIARAZIONE DI FUNZIONE

Questa dichiarazione ha la seguente forma:

NN DEF   FNK (X) = Espressione

- NN è il numero della linea.
- FNK è il nome della funzione definita da chi programma: esso deve cominciare per FN ed essere seguito dal nome di un identificatore BASIC, ma non deve assolutamente essere una parola riservata del linguaggio.
- X è una variabile simbolica detta muta in quanto la definizione della funzione di questa variabile non è obbligatoriamente conosciuta. Essa rappresenta la variabile libera della funzione e può essere sostituita da qualunque valore durante l'utilizzazione della funzione stessa. Viene quindi associata a una variabile X0 o a una costante del programma che sostituisce simbolicamente X con un valore nell'espressione che definisce questa funzione.
- L'espressione che definisce la funzione è una espressione che si riferisce agli operatori o alle funzioni standard già definite nel programma.
- Questa funzione deve certamente essere una funzione della variabile muta o libera definita qui sopra.

Nota. – Su alcuni BASIC, è possibile definire alcune funzioni che lavorano su variabili stringhe di caratteri.

Esempi:

Si debba definire un polinomio di terzo grado del tipo:

$$Ax^3 + Bx^2 + Cx + D$$

Si utilizzerà la dichiarazione:

```
10 DEF FNPO (X) = A * X ↑ 3 + B * X ↑ 2 + C * X + D
```

```
 5      INPUT A, B, C, D
10      DEF FNPO (X) = A * X↑3 + B * X↑2 + C * X + D
20      Y = FNPO (1)
30      FOR I = 0 TO 10
40      PRINT FNPO (I);
50      NEXT I
60      END
```

Esecuzione

```
?    1, 2, 3, 4 ®
□    4   10   26   58   112   194   310   466   668   922   1234
```

— È possibile anche definire delle funzioni che possono essere ottenute partendo da funzioni standard.

Esempi:

— Se vogliamo ottenere le funzioni trigonometriche utilizzando un angolo in gradi, si definirà:

```
10      DEF FNCO (X) = COS (X * π/180)
20      DEF FNSI (X) = SIN (X * π/180)
30      DEF FNTA (X) = TAN (X * π/180)
40      FOR I = 0 TO 60 STEP 15
50      PRINT I; FNCO (I); FNSI (I); FNTA (I)
60      NEXT I
70      END
```

Questo programma permette di stampare i valori del COS, SIN, TAN degli angoli in gradi da 0 a 60 con un passo di 15°.

Esecuzione:

		COS	SIN	TAN
	0	1	0	0
Angoli	15	0.965925826	0.258819045	0.26949192
in	30	0.866025404	0.5	0.577350269
gradi	45	0.707106782	0.707106781	0.999999999
	60	0.5	0.866025404	1.732205081

— Funzione Logaritmo decimale

Oltre a quello che abbiamo già visto nel paragrafo precedente, possiamo anche definire:

$$\text{Log}_{10}(x) = \text{Log}\ x/\text{Log}\ 10$$

Il programma seguente stampa i valori di Log 10 da 1 a 10.

```
10        DEF FNLO (X) = LOG (X) / LOG (10)
20        FOR I = 1 TO 10
30        PRINT I; FNLO (I);
40        NEXT I
50        END
```

Esecuzione:

```
 1        0
 2        0.30102996
 3        0.477121255
 4        0.60259991
 5        0.698970004
 6        0.77815125
 7        0.84509805
 8        0.903089987
 9        0.954242509
10        1
```

ALTRE FUNZIONI MATEMATICHE DERIVATE DA FUNZIONI STANDARD

a) *Funzioni trigonometriche inverse:*

Arc sin (x): FNAS (X) = ATN (X/SQR (– X * X + 1))
Arc cos (x): FNAC (X) = – ATN (X/SQR (– X * X + 1)) + π/2
Arc cotg (x): FNAT (X) = – ATN (X) + π/2

b) *Funzioni iperboliche:*

cos h (x): FNCH (X) = (EXP (X) + EXP (– X))/2
sin h (x): FNSH (X) = (EXP (X) – EXP (– X))/2
tang h (x): FNTH (X) = – EXP (– X)/(EXP (X) + EXP (– X)) * 2 + 1
cotang h (x): FNGH (X) = EXP (– X)/(EXP (X) – EXP (– X)) * 2 + 1

c) *Funzioni iperboliche inverse:*

Arc sin h (x): FNHS (X) = LOG (X + SQR (X * X + 1))
Arc cos h (x): FNHC (X) = LOG (X + SQR (X * X – 1))
Arc tan h (x): FNHT (X) = LOG ((1 + X)/(1 – X))/2
Arc cotg h (x): FNHG (X) = LOG ((X + 1)/(X – 1))/2

Applicazioni: Calcolo di un numero arrotondato ad un certo numero di decimali

Negli esempi dati sino ad ora, i numeri decimali erano espressi con otto cifre decimali. Nella maggior parte dei casi, ciò è superfluo. Per arrotondare un numero a n decimali, è sufficiente moltiplicarlo per 10^n, prenderne la parte intera e dividere il numero ottenuto per 10^n.

Esempio:

Si debba arrotondare a due decimali il numero 145.7895678.

Moltiplicandolo per $10^2 = 100$, si ottiene 14578, dividendo questo per $10^2 = 100$, si ottiene 145.78.

Se comunque si desidera tener conto del decimale seguente (in questo caso 9), si può aggiungere 0.5 al numero ottenuto dopo la moltiplicazione per 100. Nel nostro caso avremo 14579.45678. Ciò darà finalmente il numero arrotondato alla cifra decimale superiore: 145.79. Se il decimale seguente fosse stato tra 0 e 4, si sarebbe ottenuto un arrotondamento dell'ultima cifra al decimale inferiore.

Programma. — Prima di tutto, definiamo una funzione AR (X) tale che:

AR (X) = INT (X * 10 ↑ N + 0.5)/10 ↑ N

Il programma è il seguente:

```
10      DEF FNAR (X) = INT (X * 10↑N + 0.5)/10↑N
20      PRINT "NUMERO DEI DECIMALI"; N
30      INPUT X
40      PRINT X; "ARROTONDATO A"; N; "DECIMALI ="
45      PRINT FNAR (X)
```

```
50        GO TO 20
60        END
```

Esecuzione:

```
    NUMERO DI DECIMALI ? 3 ®
?   45.897895 ®
□   45.897895 ARROTONDATO A 3 DECIMALI = 45.897
    NUMERO DI DECIMALI ? 2 ®
?   0.123456 ®
□   0.123456 ARROTONDATO A 2 DECIMALI = 0.12
    NUMERO DI DECIMALI ? 0 ®
?   456.125 ®
□   456.125 ARROTONDATO A 0 DECIMALI = 456
```

Nota. — Una applicazione del genere è utilissima, in particolare per alcune applicazioni di gestione in cui si vogliano arrotondare i risultati definitivi del calcolo alle migliaia di lire.

Trovare uno zero reale di un polinomio di terzo grado:

Un polinomio del tipo

$$y = f(x) = a_n x^n + ... + a_2 x^2 + a_1 x + a_0$$

avrà uno zero reale compreso tra x_1 e x_2 se

$$f(x_1) \times f(x_2) < 0$$

Infatti, poichè si tratta di una funzione continua, $f(x_1)$ e $f(x_2)$ sono di segno opposto, quindi la funzione, tra i due punti x_1 e x_2, passa per uno zero. Per fare ciò, si farà una prima ricerca all'interno dell'intervallo x_1, x_2, in cui si è certi essere lo zero, successivamente si farà la stessa cosa in un intervallo 10^2 volte più piccolo e così via. Una soluzione è data dal seguente programma:

```
  1       INPUT A, B, C, D
  5       DEF FNAR (X) = (X * 100 + 0.5)/100
 10       DEF FNP (X) = A * X↑3 + B * X↑2 + C * X + D
 20       D1 = 1 : D2 = 0.01
 30       X1 = − 10 : X2 = 10
 40       FOR X = X1 TO X2 STEP D1
 50       IF FNP (X) * FNP (X + D1) < = 0 THEN 100
 60       NEXT X
 70       IF D1 = D2 THEN 200
```

```
80        D1 = D1/10
90        GO TO 40
100       IF D1 = D2 THEN I = X : GO TO 140
110       FOR I = X TO X + D1 STEP D2
120       IF FNP (I) * FNP (I + D2) < 0 THEN 140
130       NEXT I
140       R = FNAR (I)
150       PRINT "RADICE ="; R
160       GO TO 1
200       PRINT "NESSUNA RADICE NELL'INTERVALLO"; X1; X2
210       GO TO 1
220       END
```

Esecuzione:

?	3, 7, 1, – 2 ®	$(3x^3 + 7x^2 + 1x - 2)$
	RADICE = – 2.00	
?	– 4, 4, 3, 4 ®	$(-4x^3 + 4x^2 + 3x + 4)$
	RADICE = 1.75	
?	0, 1, 1, – 2 ®	$(x^2 + x - 2)$
	RADICE = – 2	

Le funzioni di più istruzioni

Alcuni BASIC permettono di definire alcune funzioni di più parametri associati a un calcolo che necessita di più istruzioni.

In questo caso la funzione è definita dal nome della funzione associata a un elenco di parametri. Il termine della definizione di funzione è indicato da una istruzione di fine.

Il ritorno all'istruzione da cui si è partiti viene effettuato, come per i sottoprogrammi, mediante l'istruzione RETURN.

La struttura del programma è allora:

```
DEF       FN   nome della funzione (elenco dei parametri)
Istruzioni contenenti almeno una istruzione
RETURN parametro
END       FN
```

Una simile struttura non è standardizzata, infatti alcuni BASIC ne danno una diversa formulazione, ad esempio:

```
DEF       FN   identificatore (p1, p2, ......pn)
corpo delle funzioni con una istruzione
RETURN pi
FNEND
```

I $p1...pn$ sono parametri trasmessi alla funzione.

Nota. — Una simile struttura è assai apprezzata da coloro che conoscono altri linguaggi di programmazione, in quanto colma una delle lacune essenziali del linguaggio BASIC standard, permettendo di definire blocchi di istruzioni richiamabili a partire da un programma, trasmettendo solo i parametri necessari al calcolo.

Esercizi

1. *Definire una funzione che calcoli:*

$$SL(A) = 2.549 \log (A + A^2 + \frac{1}{A})$$

dove log rappresenta il logaritmo decimale.

2. *Utilizzare la funzione dell'esercizio 1 per calcolare:*

$$R = X + \log X + 2.549 \log (X + X^2 + \frac{1}{X})$$

Soluzioni

1.
```
10     DEF FNSL (A) = 2.549 * LOG (A + A * A + 1/A)/LOG (10)
```
2.
```
 1     INPUT X
10     DEF FNLO (X) = LOG (X)/LOG (10)
20     DEF FNSL (A) = 2.549 * FNLO (A + A * A + 1/A)
30     R = X + FNLO (X) + FNSL (A)
40     PRINT FNLO (X); FNSL (X); R
50     END
```

Esecuzione

```
?  2 ®
   .301029996     2.0711615     4.37314614
?  3 ®
   .4777121235    2.78116412    6.25828537
```

3-3. I Sottoprogrammi

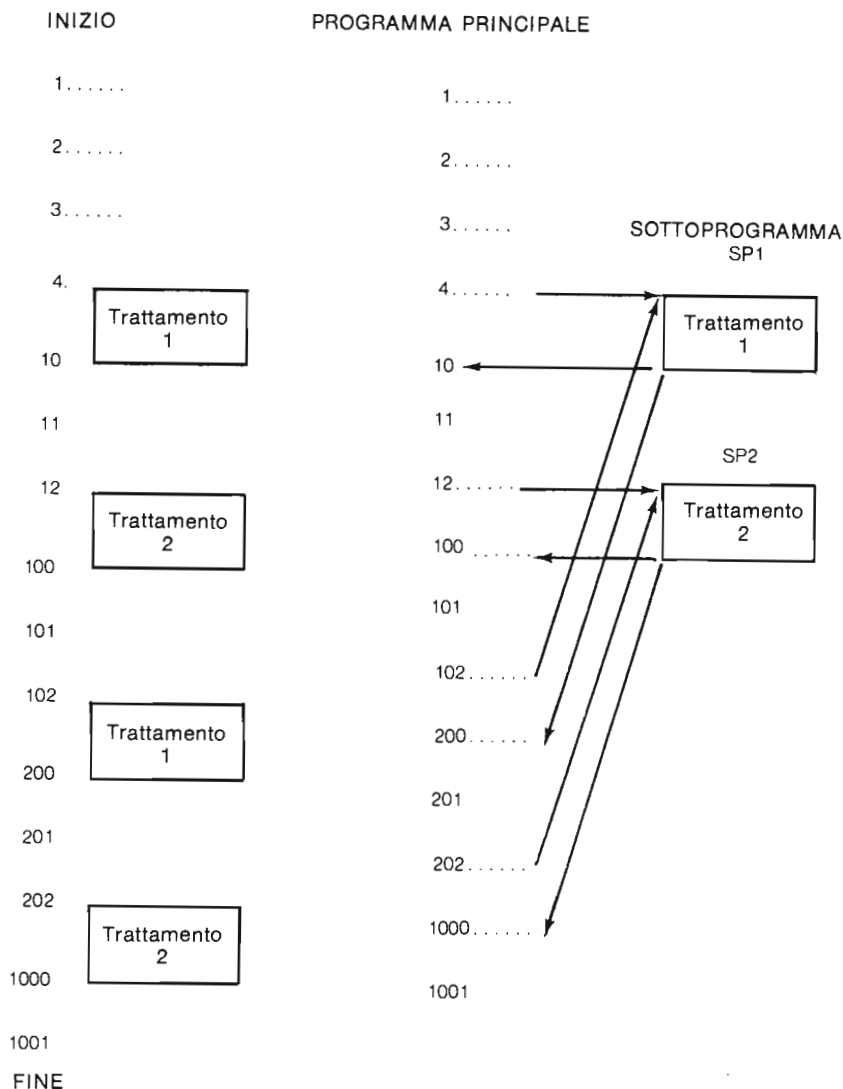
In alcuni programmi lo stesso trattamento può essere effettuato più volte a diversi indirizzi di programma. La definizione di funzione permette di far più volte riferimento allo stesso trattamento, se questo può essere espresso con una espressione aritmetica. Tuttavia, se questo trattamento necessita di più istruzioni, le funzioni non sono più utilizzabili. Bisogna allora far ricorso alla nozione di sottoprogramma (Subroutine in inglese).

DEFINIZIONE

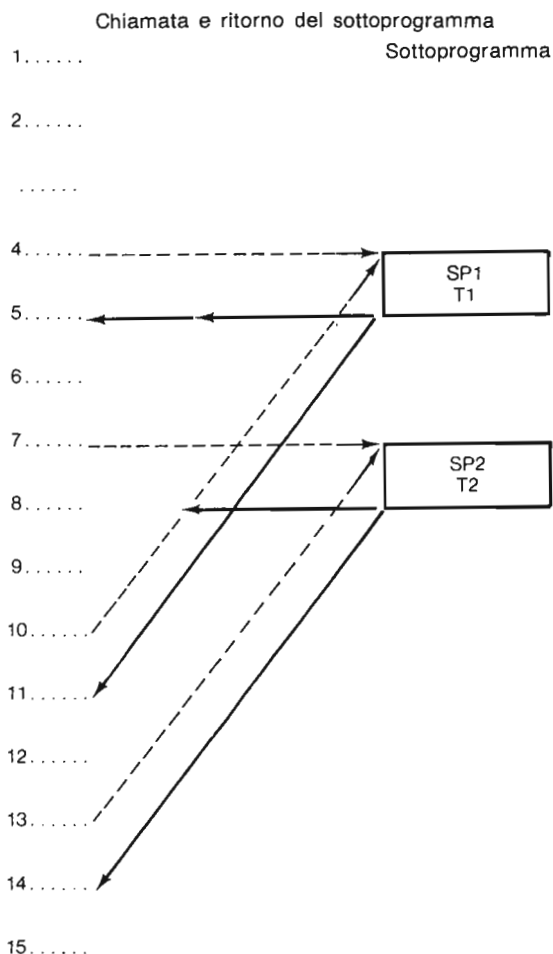
Un sottoprogramma è una serie di istruzioni realizzante un trattamento che può essere richiamato a partire da un altro programma o sottoprogramma, e che termina con una istruzione di RETURN al programma che l'ha richiamato.

L'utilità di un sottoprogramma è doppia:

- Permette di evitare il moltiplicarsi delle istruzioni che effettuano un certo trattamento, quando questo è necessario in più indirizzi del programma.
- Permette di riutilizzare alcune parti di programmi già verificate e necessarie ad altri programmi. Permette poi di costruire dei nuovi programmi con moduli già «prefabbricati». Benchè non sia il caso dei BASIC standard, un sottoprogramma può essere considerato come una «scatola nera» nella quale entrano dei dati e dalla quale escono dei risultati, senza che ci si debba preoccupare di ciò che si trova all'interno del sottoprogramma.

Supponiamo che un programma si presenti nella seguente maniera:
Si vuole che i trattamenti 1 e 2 si ripetano due volte nel programma. Trattamento 1 e Trattamento 2 possono essere considerati come dei sottoprogrammi.
Così lo schema sulla destra rappresenta la nuova struttura: esiste quindi una sola versione delle istruzioni che compongono i trattamenti 1 e 2.

Esempio di struttura di sottoprogramma

All'istruzione 4 si ha un salto verso il sottoprogramma SP1 e, appena il trattamento è terminato il ritorno si effettua al 10, che è l'istruzione che segue immediatamente l'istruzione di salto. Lo stesso procedimento è utilizzato all'istruzione 202, con un ritorno al 1000, dopo l'esecuzione dello stesso sottoprogramma SP1.

In pratica, le istruzioni del programma principale (cioè quello che chiama il sottoprogramma) potrebbero essere numerate così:

Così si può enunciare questa *regola*:

Il ritorno di un sottoprogramma si effettua sempre verso l'istruzione che segue l'istruzione di chiamata o di salto al sottoprogramma.

LE ISTRUZIONI DI CHIAMATA E DI RITORNO DI UN SOTTOPROGRAMMA

Sono assai semplici.

La chiamata o il salto ad un sottoprogramma è:

– CHIAMA SP NN o VAI SP NN

(SP = sottoprogramma), NN è il numero della prima istruzione del sottoprogramma.

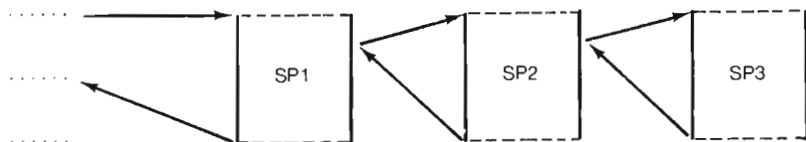
In BASIC, questa istruzione è:
– GOSUB NN

Il ritorno da un sottoprogramma è semplicemente una istruzione:
TORNA (RETURN).

In questo caso, non è necessario, a causa della regola prima definita, precisare a quale numero di istruzione si torna.

Nota. — Si possono avere più punti di entrata in un sottoprogramma.
- Si possono avere più istruzioni di ritorno nello stesso sottoprogramma.
- Benchè nel BASIC sia possibile uscire da un sottoprogramma con una istruzione VAI A, senza passare da una istruzione TORNA, ciò è una *PESSIMA tecnica* di programmazione. In principio, si deve uscire da un sottoprogramma solo attraverso una istruzione TORNA (RETURN).

L'INCASTRO DEI SOTTOPROGRAMMI

È possibile chiamare un secondo sottoprogramma, partendo da un primo. Si può anche avere l'incastro di più livelli di sottoprogrammi.

Programma principale e struttura dei sottoprogrammi

Qui esistono tre livelli di sottoprogrammi.

Il programma principale chiama SP1, che chiama SP2, che chiama SP3.

Si è stabilito che, nelle strutture di questo tipo, esista un limite nel numero dei livelli: in pratica questo limite dipende dall'interprete, nella maggior parte dei sistemi non lo si ha.

Si abbia un programma che studia due parametri quantitativi concernenti una popolazione di individui (ad esempio, il peso e la statura).

Si tratta di scrivere un programma che calcoli la media, la varianza e lo scarto tipo di questi due parametri.

Ricordiamo che si hanno *n* individui, considerando che x_i rappresenta la misura dell'individuo *i*, la media, per definizione è:

$$m = \frac{\sum_{i=1}^{n} x_i}{n}$$

e la varianza teorica è:

$$V = \frac{\sum_{i=1}^{n} x_i^2}{n} - m^2$$

la sua stima è:

$$V = \frac{\Sigma x^2 - (\Sigma x)^2/n}{n - 1}$$

mentre lo scarto tipo o deviazione standard è

$$D = \sqrt{V}$$

In questo caso, è evidente che si deve considerare il calcolo della media e della varianza come un sottoprogramma che sarà chiamato due volte: una per il primo parametro e una per il secondo.

Incominciamo quindi scrivendo il sottoprogramma.

Abbiamo già visto il calcolo della media. Per calcolare le varianze basta calcolare la somma dei quadrati Σx_i^2.

Se n è il numero degli elementi si ha dunque un sottoprogramma del tipo:

```
200     S = 0
210     V = 0
220     D = 0
230     FOR I = 1 TO N
240     S = S + X (I)
250     V = V + X (I) * X (I)
260     NEXT I
270     M = S/N
280     V = V/N − M↑2        oppure V = (V − S↑2/N)/(N − 1)
290     D = SQR (V)
300     RETURN
```

Il programma principale è allora incaricato di leggere i dati e di stampare i risultati. Una prima soluzione consiste nel leggere i dati successivamente, utilizzando la tabella X.

```
 10      INPUT N
 20      DIM X (N)
 30      REM LETTURA DEL PRIMO PARAMETRO
 35      PRINT "PESI"
 40      FOR I = 1 TO N
 50      INPUT X (I)
 60      NEXT I
 70      GOSUB 200
 90      PRINT "MEDIA"; M; "VARIANZA"; V; "SCARTO"; D
100      REM LETTURA DEL SECONDO PARAMETRO
105      PRINT "STATURA"
110      FOR I = 1 TO N
120      INPUT X (I)
130      NEXT I
140      GOSUB 200
160      PRINT "MEDIA"; M; "VARIANZA"; V; "SCARTO"; D
170      END
```

Esecuzione

```
?   5 ®numero di individui
PESI
?   65
?   70
?   80
?   75
?   72
□   MEDIA 72.4   VARIANZA 25.0399931   SCARTO 5.00399742
STATURA
?   1.60
?   1.75
?   1.80
?   1.70
?   1.65
□   MEDIA 1.7   VARIANZA 4.99999766 E—03
SCARTO 0.07071606
```

Seconda soluzione

Nell'esempio precedente, si aveva la lettura successiva dei dati in una tabella. In pratica, se non interessa conservare i dati, si può superare l'uso della tabella. Si ottiene allora il programma seguente. In questo esempio, si utiliz-

zano due sottoprogrammi, il primo per leggere i dati e accumulare le somme parziali Σx e Σx^2. Il secondo calcola M, V, D e li stampa.

```
10          INPUT N
20          PRINT "PESI"
30          GOSUB 100
40          GOSUB 200
50          PRINT "STATURA"
60          GOSUB 100
70          GOSUB 200
80          END
```

Primo sottoprogramma

```
100         S = 0 : V = 0 : D = 0
110         FOR I = 1 TO N
120         INPUT X
130         S = S + X
140         V = V + X * X
150         NEXT I
160         RETURN
```

Secondo sottoprogramma

```
200         M = S/N
210         V = V/N - M * M    oppure    V = (V - S * S/N)/(N - 1)
220         D = SQR (V)
230         PRINT "MEDIA"; M; "VARIANZA"; V;
240         PRINT "SCARTO"; D
250         RETURN
```

Esecuzione

```
?     5 ®
PESI
?     60
?     55
?     45
?     65
?     50
□     MEDIA 55    VARIANZA 49.9999979    SCARTO 7.07106767
STATURA
?     1.62
?     1.50
```

```
?   1.45
?   1.60
?   1.55
□   MEDIA 1.544    VARIANZA 3.94399777 E–03
SCARTO 0.06280126
```

Terza soluzione

Si supponga che si vogliano conservare i dati di entrata. In questo caso bisogna ritornare alla prima soluzione e utilizzare due tabelle P (Peso) e A (altezza = statura) per leggere i dati e una tabella X per fare i calcoli.

```
10    INPUT N
20    DIM P (N), A (N), X (N)
30    FOR I = 1 TO N
40    INPUT P (I), A (I)
50    NEXT I
60    PRINT "PESI"
70    FOR I = 1 TO N
80    X (I) = P (I)
90    NEXT I
100   GOSUB 200
110   GOSUB 300
120   PRINT "STATURA"
130   FOR I = 1 TO N
140   X (I) = A (I)
150   NEXT I
160   GOSUB 200
170   GOSUB 300
180   END
190   REM PRIMO SOTTOPROGRAMMA
200   S = 0, V = 0, D = 0
210   FOR I = 1 TO N
220   S = S + X (I)
230   V = V + X (I) * X (I)
240   NEXT I
250   RETURN
```

Il secondo sottoprogramma è identico a quello della seconda soluzione:

```
300   M = S / N
310   V = V / N – M * M oppure V = (V – S ↑ 2/N) / (N – 1)
320   D = SQR (V)
330   PRINT "MEDIA"; M; "VARIANZA"; V
340   PRINT "SCARTO"; D
350   RETURN
```

Esecuzione

```
?   5 ®
?   56, 1.6 ®
?   60, 1.62 ®
?   65, 1.70 ®
?   70, 1.75 ®
?   72, 1.78 ®
□   PESI
    MEDIA 64.6    VARIANZA    35.83      SCARTO 5.98
□   STATURA
    MEDIA 1.69    VARIANZA    4.959E-03    SCARTO 0.0704
```

Nota sui sottoprogrammi in BASIC

Gli esempi precedenti illustrano la difficoltà di fare dei sottoprogrammi generali in BASIC. Infatti, poiché non c'è trasmissione dei parametri tra il programma che richiama e il proprio sottoprogramma, bisogna che le variabili utilizzate nel sottoprogramma siano conosciute dal programma principale e viceversa. Questo è uno degli inconvenienti maggiori del linguaggio: *non è possibile sviluppare dei sottoprogrammi parametrati*; ciò riduce la flessibilità e l'utilità dei sottoprogrammi nel BASIC. Bisogna tuttavia notare che su alcune realizzazioni di BASIC la definizione di sottoprogrammi parametrati è possibile.

Termineremo questo paragrafo, presentando un programma più complesso, dove la scomposizione in sottoprogrammi permette di capire meglio, di risolvere e di programmare il problema posto.

Il gioco di NIM

Il gioco di NIM, detto gioco di Marienbad, è giocato qui con dodici fiammiferi disposti su tre file, nella maniera seguente:

```
I   I   I
I   I   I   I
I   I   I   I   I
```

Il gioco consiste nel togliere dei fiammiferi seguendo le regole riportate qui sotto. L'ultimo a ritirare uno o più fiammiferi ha vinto.

L'insieme è costituito da n_1 fiammiferi nella prima riga, n_2 nella seconda, n_3 nella terza e può essere rappresentato dalla terna (n_1, n_2, n_3). La situazione iniziale del gioco è allora rappresentata dalla terna (3, 4, 5). Si gioca in due. Ciascun giocatore, quando è il suo turno, può prendere uno o più fiammiferi da una riga. L'ultimo a prendere i fiammiferi vince (situazione 0, 0, 0).

Si abbiano due giocatori X e Y. La situazione (n_1, n_2, n_3) raggiunta da X è detta vincente, se, qualunque sia la mossa di Y, esiste una mossa di X che condurrà alla situazione (0, 0, 0). Quando il giocatore X ha raggiunto una posizione vincente, può procedere così in maniera sistematica, reagendo a qualunque azione del suo avversario Y con una azione che conduce di nuovo ad una situazione vincente. Questa strategia permette al giocatore X di aspettare eventualmente la situazione (0, 0, 0) e di vincere la partita. La conoscenza delle situazioni vincenti è dunque importantissima.

Ora, è possibile determinare, nel caso del gioco di NIM, se una situazione è vincente o no, con il metodo seguente (questo algoritmo può essere generalizzato ad un numero qualunque di fiammiferi, di righe, e ad una disposizione iniziale qualunque. Provate):

1. Sia $r_1 = b_1 + c_1 + d_1$
   $r_2 = b_2 + c_2 + d_2$
   $r_3 = b_3 + c_3 + d_3$
   dove b_1, b_2, b_3 sono gli equivalenti binari di n_1
   $c_1, c_2, c_3,$ sono gli equivalenti binari di n_2
   $d_1, d_2, d_3,$ sono gli equivalenti binari di n_3

2. Se r_1, r_2, r_3 sono tutti pari, allora la situazione è vincente, altrimenti no.
   Ad esempio (3, 4, 5) non è una situazione vincente.

   In binario 3 è $b_1\ b_2\ b_3 = 001$
   4 è $c_1\ c_2\ c_3 = 100$
   5 è $d_1\ d_2\ d_3 = 101$
   e $r_2 = 1 + 0 + 0$ è dispari
   Invece (1, 4, 5) è una situazione vincente.
   In binario 1 è $b_1\ b_2\ b_3 = 001$
   4 è $c_1\ c_2\ c_3 = 100$
   5 è $d_1\ d_2\ d_3 = 101$
   e $r_1 = 0 + 1 + 1 = 2$
   $r_2 = 0 + 0 + 0 + 0$ Tutti pari
   $r_3 = 1 + 0 + 1 = 2$

– Scrivere un programma che simuli le partite di NIM per un giocatore A contro un giocatore B. Il giocatore A fa la sua scelta sotto forma di L (numero di fiammiferi) da togliere sulla riga M. Un primo programma considera che il giocatore B scelga a caso tra i diversi colpi possibili. In un secondo programma, il giocatore B cerca la situazione vincente: se ciò non è possibile, il giocatore B toglie un fiammifero dalla prima riga possibile. Si verificherà che B vince tutte le volte che è lui a cominciare il gioco, in quanto parte da una posizione vincente al primo colpo.

1. *Programma per il gioco di NIM casuale*

```
 10   INPUT "NUMERO DI FILE"; N
 15   DIM A (N)
 20   FOR I = 1 TO N
 30   INPUT "NUMERO DI FIAMMIFERI"; A (I)
 40   NEXT I
 50   GOSUB 410
110   INPUT "TOCCA A TE"; L, M
120   IF L > N THEN 110
125   IF A (L) = 0 THEN 110
130   GOSUB 700
135   IF G = 1 THEN 140
136   PRINT "HAI VINTO"
137   GO TO 20
140   PRINT "TOCCA A ME"
145   L = INT (N * RND (1) + 1)
150   IF A (L) = 0 THEN 145
160   M = INT (A (L) * RND (M) + 1)
165   IF M = 0 THEN 160
170   GOSUB 700
180   IF G = 1 THEN 110
190   PRINT "HO VINTO"
200   GO TO 20
```

Sottoprogramma per la sottrazione di M fiammiferi dalla riga L

```
700   A (L) = A (L) – M
705   IF A (L) < 0 THEN A (L) = 0
710   GOSUB 410
745   G = 0
750   FOR I = 1 TO N
760   IF A (I) = 0 THEN 780
770   G = 1
780   NEXT I
790   RETURN
```

2. *Programma per il gioco del NIM con ricerca della soluzione vincente*

```
 10   INPUT "NUMERO DI FILE"; N
 15   DIM A (N), B (N, 10), R (10)
 20   FOR I = 1 TO N
 30   INPUT "NUMERO DI FIAMMIFERI"; A (I)
```

```
40    NEXT I
50    GOSUB 400
60    FOR I = 1 TO N
70    C = A (I)
80    GOSUB 500
90    NEXT I
100   GOSUB 600
110   INPUT "TOCCA A TE"; L, M
120   IF L > N THEN 110
125   IF A (L) = 0 THEN 110
130   GOSUB 700
135   IF G = 1 THEN 140
136   PRINT "HAI VINTO"
137   GO TO 20
140   PRINT "TOCCA A ME"
145   FOR J = 10 TO 0 STEP − 1
146   IF R (J) = 0 THEN 200
150   IF R (J) / 2 = INT (R (J) / 2) THEN 200
160   FOR I = 1 TO N
170   IF A (I) = 0 THEN 190
180   IF B (I, J) = 1 THEN 210
190   NEXT I
200   NEXT J
201   REM NESSUNA SOLUZIONE VINCENTE
202   FOR I = 1 TO N
203   IF A (I) < > 0 THEN 205
204   NEXT I
205   L = 1 : M = 1
206   GO TO 220
210   L = 1
215   GOSUB 800
220   GOSUB 700
230   IF G = 1 THEN 110
240   PRINT "HO VINTO"
250   GO TO 20
```

Sottoprogramma per la stampa

```
400   REM STAMPA DEL GIOCO
410   PRINT
420   FOR I = 1 TO N
430   IF A (I) = 0 THEN 470
440   FOR J = 1 TO A (I)
```

```
450         PRINT "I"
460         NEXT J
470         PRINT
480         NEXT I
490         RETURN
```

Sottoprogramma per la conversione decimale-binaria

```
495         REM CALCOLO DEL VALORE BINARIO DI A (I)
500         FOR J = 10 TO 0 STEP - 1
510         B (I, J) = INT (C / 2↑J)
520         C = C - B (I, J) * 2↑J
530         NEXT J
540         RETURN
```

Sottoprogramma per il calcolo di somme binarie

```
590         REM CALCOLO DEI R (I)
600         FOR J = 10 TO 0 STEP - 1
610         R (J) = 0
620         FOR J = 1 TO N
630         R (J) = R (J) + B (I, J)
640         NEXT I
650         NEXT J
660         RETURN
```

Sottoprogramma per la sottrazione di M fiammiferi da una fila L

```
700         A (L) = A (L) - M
705         IF A (L) < 0 THEN A (L) = 0
710         GOSUB 110
720         C = A (L) : I = L
730         GOSUB 500
740         GOSUB 600
745         G = 0
750         FOR I = 1 TO N
760         IF A (I) = 0 THEN 780
770         G = I
780         NEXT I
790         RETURN
```

Sottoprogramma per la ricerca di una configurazione vincente

```
800         I1 = I : J1 = J
```

```
805        FOR K = A (I) – 1 TO 0 STEP – 1
810        C = K : I = I1
820        GOSUB 500
830        GOSUB 600
840        FOR J = J1 TO 0 STEP – 1
850        IF R (J) / 2 < > INT (R (J) / 2) THEN 890
860        NEXT J
865        REM SOLUZIONE  VINCENTE
870        M = A (I1) – K
880        RETURN
890        NEXT K
895        REM NESSUNA SOLUZIONE VINCENTE
900        C = A (I1) : I = I1
910        GOSUB 500
920        GOSUB 600
930        M = 1
940        RETURN
```

4. LE ISTRUZIONI DI SCAMBIO MULTIPLO SU... VAI A (ON... GO TO)

Le istruzioni di test (SE... ALLORA) permettono di programmare qualunque algoritmo e, in particolare, gli scambi multipli. Tuttavia, in alcuni casi, è interessante poter sostituire una serie di test su una stessa variabile o espressione con una sola istruzione che permette di rinviare il controllo verso differenti zone del programma che tratta una condizione particolare. Nel BASIC esteso, questa possibilità è offerta dall'istruzione SU (ON) ...VAI A (GO TO).

Sintassi e semantica dell'istruzione

Dal punto di vista sintattico, questa istruzione si presenta sotto la seguente forma generale:

```
SU    espressione aritmetica     VAI A  lista d'etichette
ON    espressione aritmetica     GO TO  lista d'etichette
```

L'espressione aritmetica è valutata in forma intera, cioè il risultato del calcolo è troncato alla sua parte intera. La lista delle etichette è una serie di numeri separati da virgole e corrispondenti a numeri di istruzioni corrispondenti allo scambio.

Se il valore dell'espressione è uguale a 1, il controllo sarà dato all'istruzione corrispondente alla prima etichetta, se il valore dell'espressione è 2, il controllo sarà dato alla seconda..., se il valore dell'espressione è N, il controllo sarà dato alla Nª etichetta.

Infine, se il valore dell'espressione non corrisponde a un valore associato ad una etichetta della lista (valore ≤ 0 o superiore al numero di etichette), il controllo è dato all'istruzione in sequenza.

Nota. — Su alcuni BASIC, è necessario che l'espressione dia un risultato intero, altrimenti lo scambio non ha luogo e viene eseguita l'istruzione in sequenza. Altri considerano il valore negativo come un errore di esecuzione.

Esempio:

```
10      ENTRA I, N
20      SU I VAI A 40, 60, 80
30      STAMPA "ALEATORIO"; N; "="; ALE (N);
35      VAI A 90
40      STAMPA "EXP"; N; "="; EXP (N);
50      VAI A 90
60      STAMPA "LOG"; N; "="; LOG (N);
70      VAI A 90
80      STAMPA "RAD"; N; "="; RAD (N)
90      FINE
```

Questo programma chiede l'inserimento di due valori: il primo serve da variabile di controllo allo scambio, il secondo è una variabile che serve da parametro a una funzione. Se I vale 1, si stamperà il valore di e^N; se I vale 2, si stamperà il valore di Log (N); se I vale 3, si stamperà il valore di $\sqrt{N}$. Se I assume qualunque altro valore, verrà stampato un valore aleatorio. A titolo d'esempio, questo programma è stato verificato in BASIC su un microelaboratore CBM.

```
10      INPUT I, N
20      ON I GO TO 40, 60, 80
30      PRINT "ALEATORIO"; N; "="; RND (N);
35      GO TO 10
40      PRINT "EXP"; N; "="; EXP (N);
50      GO TO 10
60      PRINT "LOG"; N; "="; LOG (N);
70      GO TO 10
80      PRINT "RAD"; N; "="; SQR (N);
90      GO TO 10
100     END
```

Dal programma si sono ottenuti questi risultati:

```
?    1. 2 ®
     EXP 2 = 7.3890561
```

```
?     2, 2 ®
      LOG 2 = 0.693147181
?     3, 2 ®
      RAD 2 = 1.41421356
?     4, 5 ®
      ALEATORIO 5 = 0.652794473
?     1.4, 1 ®
      EXP 1 = 2.71828183
?     0.2, 1 ®
      ALEATORIO 1 = 0.461619015
?     2.4, 3 ®
      LOG 3 = 1.09861229
?     2.78, 9 ®
      RAD 9 = 3
?     79.45, 10 ®
      ALEATORIO 10 = 0.021016756
?     −1, 2 ®
      ILLEGAL QUANTITY ERROR IN 20
      (ERRORE QUANTITÀ NON AMMESSA IN 20)
```

Si può concludere che, per questa realizzazione di BASIC, tutte le regole enunciate sono valide. La sola eccezione è che i valori negativi dell'espressione non sono accettati. In questo caso, se non si conosce a priori il valore dell'espressione, sarà utile fare prima un test su tutte le istruzioni SU (ON), per assicurarsi che l'espressione non assuma valori negativi e per inserire un messaggio d'errore corrispondente.

Altri esempi

– Se nel programma precedente si aggiungono le istruzioni:

```
15        K = I↑2 + 1
20        ON K GO TO 40, 60, 80
```

nell'esecuzione si ottiene:

```
?     −2. 2 ®
      ALEATORIO 2 = .786648867
?     1, 2 ®
      LOG 2 = 0.693147181
?     0, 1 ®
      EXP 1 = 2.71828183
?     1.5, 2 ®
      RAD 2 = 1.41421356
```

In questo caso si possono inserire valori negativi senza errori nell'esecuzione.

– Se si sostituisce l'istruzione 15 con:

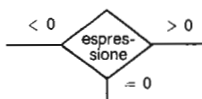
```
15        K = SGN (I) + 1
```

dove SGN è una funzione che dà il segno di I, che vale – 1 se il segno è negativo, + 1 se è positivo, 0 se il valore è 0. Si ottengono allora le esecuzioni seguenti:

```
?    – 1, 2 ®
     ALEATORIO 2 = .714959655
?    0, 2 ®
     EXP 2 = 7.3890561
?    2, 4 ®
     LOG 4 = – 1.38629436
```

In questo caso non è possibile calcolare la funzione radice quadra, poiché il valore massimo di K è $1 + 1 = 2$.

L'esempio precedente è uno scambio che permette di fare un test che riguarda tre possibilità: valore di una espressione > 0, $= 0$ e < 0.

Ciò può essere rappresentato con un test a tre rami

Negli altri linguaggi, FORTRAN in particolare, questo test è detto test aritmetico. Per quanto riguarda il caso generale di uno scambio a *n* uscite, questo non ha una rappresentazione standard, ma si può, per esempio, utilizzare lo schema seguente:

Gli scambi verso i sottoprogrammi

L'istruzione SU (ON), dove esiste, è ugualmente utilizzabile per degli scambi verso sottoprogrammi. In questo caso, la sintassi è identica a quella dell'istruzione SU, ma la parola chiave VAI A (GO TO) è sostituita da CHIAMA SP (chiama il sottoprogramma), in inglese GOSUB.

SU espressione CHIAMA SP lista di etichette
ON espressione GOSUB SP lista di etichette

L'interesse di questa istruzione sta nel fatto di poter richiamare dei sottoprogrammi differenti, a seconda del valore di una espressione o di una variabile.

Tornando da un sottoprogramma il controllo è allora dato all'istruzione che segue l'istruzione SU (ON).

Poichè, d'altra parte, questa istruzione è quella che viene eseguita se l'espressione non corrisponde a una delle etichette, bisogna dunque assicurarsi che il trattamento desiderato si effettui nel caso in cui si ha una espressione che dà un valore che non corrisponde all'esecuzione del sottoprogramma.

La recursività nel BASIC

Alcune realizzazioni del BASIC permettono di definire dei programmi recursivi. È il caso specialmente delle funzioni multilinee sul BASIC sviluppato da ZILOG. Espressa in maniera semplice, la recursività permette di definire una funzione attraverso una espressione, o una serie di istruzioni, che utilizza essa stessa.

Così, la funzione fattoriale può essere definita recursivamente da:

$$FAC (N) = FAC (N - 1) \times N$$

Allo stesso modo, per risolvere il problema dell'inversione di una stringa di caratteri C$, si può utilizzare la funzione multilinea seguente:

```
DEF FNI$ (C$)
IF LEN (C$) < = 1 THEN RETURN C$
RETURN FNI$ (C$ [2]) + C$ [1, 1]
```

Questa funzione inverte subito il secondo carattere della stringa C$, al quale si aggiunge per concatenamento il primo carattere della stringa stessa. Se si avesse C$ = ABCDE, la prima iterazione porta BCDEA e alla fine si ottiene EDCBA.

5. UN ESERCIZIO DI STILE: IL PROBLEMA DELLE OTTO REGINE

Per terminare questo capitolo, studieremo un problema un po' più difficile. Nel gioco degli scacchi, è possibile piazzare otto regine su una scacchiera, senza che ciascuna di queste sia minacciata da un'altra. Ricordiamo che, nel gioco degli scacchi, una regina controlla tutte le posizioni che si trovano sulla stessa linea, sulla stessa colonna e sulle due diagonali che passano per la posizione della regina stessa. Questo problema ha più di una soluzione e, per molto tempo, fu l'oggetto di ricerca sia da parte di giocatori di scacchi, sia da parte di matematici. Così, alla metà del XIX secolo, se ne erano già trovate 40 soluzioni.

Il matematico Gauss riteneva che le soluzioni fossero 76. Oggi si sa che le soluzioni sono 92, ma questo valore è stato ottenuto semplicemente contandole, infatti non abbiamo conoscenza di una dimostrazione matematica che

lo provi. Ci si propone quindi di scrivere un programma che ricavi 92 soluzioni.

Consigliamo il lettore di cercare, innanzitutto, un algoritmo che permetta di ottenere questo risultato. Benchè il programma non sia affatto lunghissimo, riteniamo che si tratti di un problema assai complesso, è quindi necessario farne un'analisi assai dettagliata.

Definizione di una posizione senza conflitto

Non è certo pensabile, nemmeno con un elaboratore, di valutare tutte le configurazioni possibili della disposizione di otto regine su una scacchiera (addirittura, mettendo solamente una regina per colonna, si avrebbero più di 10^9 possibilità, tra le quali si dovrebbero ricercare le soluzioni valide!).

Bisogna dunque cercare una strategia per piazzare le regine, tenendo conto delle regine già piazzate. È evidente infatti che non si può piazzare una regina sulla colonna o sulla riga in cui è già piazzata un'altra regina. L'algoritmo di inserimento delle regine deve dunque tener presente la necessità di non mettere più di una regina per colonna. Successivamente, all'interno di ciascuna colonna, l'algoritmo dovrà ricercare la o le righe più convenienti. Infine, prima di piazzare la regina su una riga, bisognerà assicurarsi che non sia minacciata da una regina sulle diagonali!

Una posizione di tal genere si chiama *posizione senza conflitto*: è una posizione che, tenendo conto delle regine già piazzate, permette di piazzare una nuova regina, senza che questa sia minacciata da quelle già in gioco.

Principio dell'algoritmo

Quando si è arrivati all'ottava colonna, trovando una posizione senza conflitto, si ha dunque una soluzione al problema.

Però bisogna trovarle tutte. Per fare ciò, bisogna utilizzare un algoritmo di ritorno all'indietro («backtraking»). Questo algoritmo consiste nel ritornare alla colonna precedente, quando si è trovata una soluzione o quando non la si è trovata per una colonna qualunque, ad un certo momento dell'algoritmo. Ciò permette di verificare se esiste un'altra soluzione senza conflitto, e di ripartire, avanzando fino all'ultima colonna o fino ad una impossibilità. Quando questa operazione è stata ripetuta fino alla colonna 1, significa che non si hanno soluzioni per quella posizione di regina nella colonna 1. Si cambia allora la posizione della regina nella colonna 1 e si ripete il processo, fin quando la regina della colonna 1 è stata piazzata sulle otto linee di quella colonna.

L'algoritmo può allora essere riassunto secondo i seguenti punti:

1. Piazzare la prima regina sulla colonna 1 ($C = 1$)
2. Se $C \geq 8$ stampa la soluzione e vai al punto 5.

3. Piazza la regina sulla linea 1 della colonna C (L = 1)
4. Se si ha una posizione senza conflitto, si passa alla colonna seguente (C = C + 1) e si va al punto 2.
5. Se la regina della colonna C è sulla linea 8 (L = 8), si passa alla colonna precedente C = C − 1.
6. Se C = 1 e L = 8 si è terminato.
7. Se L > 8 si va al punto 5.
   Altrimenti si va al 4.

Programmazione

La programmazione differisce un po' dall'algoritmo precedente, in quanto si utilizza una tabella CO per rappresentare la posizione di una regina in una colonna: così CO (I) rappresenta la *i*-esima colonna e il suo valore rappresenta la linea in cui si trova la regina in quella colonna. (Se CO (I) = 3, la regina della *i*-esima colonna si trova nella terza riga). Per verificare l'assenza di conflitto sulle diagonali, basta verificare che le regine non sono sulle rette di pendenza + 1 o − 1.

Se si hanno due regine in posizione $x1$, $y1$ e $x2$, $y2$, è sufficiente valutare che:

$$\left|\frac{y1 - y2}{x1 - x2}\right| \neq 1$$

e cioè $| y1 - y2 | - | x1 - x2 | \neq 0$
Ciò viene effettuato con l'istruzione 110.

Per la stampa dei risultati una casella che contiene la regina è indicata con 1, altrimenti con 0.

Programma che permette di ottenere tutte le configurazioni di 8 regine su una scacchiera

```
 10    DIM CO (8), E (8)
 20    NS = 1
 40    FOR I = 1 TO 8
 50    CO (1) = I
 60    C = I
 65    L1 = 1
 70    FOR L = L1 TO 8
 75    REM VERIFICARE CHE NON CI SIANO CONFLITTI
 80    FOR K = 1 TO C
 90    DL = ABS (CO (K) – L)
100    IF DL = 0 THEN 140
```

```
110         IF DL = ABS (C + 1 – K) THEN 140
120         NEXT K
130         CO (C + 1) = L : GO TO 200
140         NEXT L
145         REM INDIETREGGIARE DI UNA COLONNA
150         C = C – 1
160         IF C = 0 THEN 300
170         L1 = CO (C + 1) + 1
180         IF L1 > 8 THEN 150
190         GO TO 70
195         REM AVANZARE DI UNA COLONNA
200         C = C + 1
205         IF C < = 7 THEN 65
206         PRINT "SOLUZIONE"; NS
207         REM STAMPA DI UNA SOLUZIONE
210         FOR J = 1 TO 8
220         FOR K = 1 TO 8
230         E (K) = 0
240         E (CO (J)) = 1
250         PRINT E (K);
260         NEXT K
270         PRINT
280         NEXT J
290         NS = NS + 1
295         GO TO 150
300         NEXT I
```

Esecuzione:

Diamo qui semplicemente le prime soluzioni; se il programma è corretto, se ne stamperanno 92.

Soluzione 1

```
1  0  0  0  0  0  0  0
0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  1
0  0  0  0  0  1  0  0
0  0  1  0  0  0  0  0
0  0  0  0  0  0  1  0
0  1  0  0  0  0  0  0
0  0  0  1  0  0  0  0
```

Soluzione 2

```
1  0  0  0  0  0  0  0
0  0  0  0  0  1  0  0
0  0  0  0  0  0  0  1
0  0  1  0  0  0  0  0
0  0  0  0  0  0  1  0
0  0  0  1  0  0  0  0
0  1  0  0  0  0  0  0
0  0  0  0  1  0  0  0
```

Soluzione 3

1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0

Soluzione 4

1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0

Soluzione 5

0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0

Soluzione 6

0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0

L'ultima soluzione è
Soluzione 92

0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0

CONCLUSIONE

In questo capitolo abbiamo presentato le ultime caratteristiche essenziali del linguaggio BASIC. Chi lo utilizzerà deve, a questo punto, essere in grado di prendere in considerazione la programmazione di qualunque problema che può essere risolto con metodi algoritmici.

Abbiamo visto in particolare il modo di trattare dati strutturati in liste o tabelle, oltre alle possibilità di utilizzare e di definire alcune nuove funzioni matematiche.

Infine, abbiamo presentato e studiato la nozione di sottoprogramma. Tutte queste nozioni, d'altronde, sono comuni ad ogni linguaggio evoluto. Bisogna notare, tuttavia, una certa carenza nel linguaggio BASIC standard, per quan-

to riguarda la definizione di sottoprogrammi parametrati. Questa carenza è evitata, alcune volte, su certi derivati del BASIC che introducono le nozioni di variabili globali o locali, che qui non abbiamo sviluppato.

Le nozioni che svilupperemo nel capitolo seguente sono più specifiche e non sono in generale completamente standardizzate.

Proponiamo, alla fine di questo capitolo, un certo numero di esercizi o di problemi senza darne la soluzione.

Esercizi (di cui non è data la soluzione)

1. *Il sistema di numerazione romana utilizza i simboli M, D, C, L, X, V, I, per rappresentare i numerii 1000, 500, 100, 50, 10, 5, 1,*
   In latino «arcaico» un numero veniva rappresentato con una sequenza di questi simboli, scritti da sinistra verso destra, e il valore del numero era la somma dei valori associati a ciascun simbolo. Nel latino classico, i simboli C, X, I possono precedere un simbolo di valore più grande e, in questo caso, il simbolo C, X, I deve essere detratto dal valore corrispondente.
   Esempi:

Decimale	latino arcaico	latino classico
4	IIII	IV
9	VIIII	IX
91	LXXXXI	XCI

   a) *Scrivere un programma che legga un numero in latino arcaico e che stampi il valore decimale corrispondente. Fare l'operazione inversa (decimale — latino arcaico).*
   b) *Eseguire una addizione in latino arcaico, senza passare dal valore decimale, e stampare il risultato in latino arcaico.*
   c) *Fare la stessa operazione, passando dal valore decimale, e stampando il risultato in latino arcaico.*

2. *Scrivere un programma che realizzi le seguenti operazioni:*
   a) *Lettura in latino classico, conversione e stampa in decimale.*
   b) *Lettura di un numero decimale, stampa in latino classico.*
   c) *Realizzare una addizione, una sottrazione e una moltiplicazione di due numeri in latino classico e stampare i risultati in latino classico.*

3. *Scrivere un programma che legga un numero in decimale e lo stampi in lettere.*
   Esempio: 1291 = «Milleduecentonovantuno»
   Si verificherà il programma, in modo tale che possa funzionare per numeri fino a sette cifre.

4. *Scrivere un programma che legga un numero in lettere e lo stampi in decimali (operazione inversa alla precedente).*

5. *Scrivere un programma che stampi un calendario, tenendo conto degli anni bisestili e del fatto che gli anni all'inizio del secolo (1800, 1900, 2000) non lo sono. Si domanda anche la stampa dei giorni in lettere. Si prenderanno come esempio gli anni 1978, 1980, 2000.*

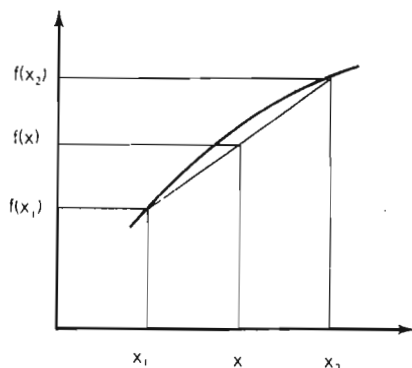
6. a) *Scrivere un programma che calcoli x^2 per:*

$$x = 1,\ 1.1,\ 1.2,\ 1.3,\ \ldots,\ 9.8,\ 9.9,\ 10$$

i valori x saranno inseriti in una tabella X e i valori x^2 in una tabella X2.

b) *Utilizzando un metodo di interpolazione lineare, scrivere il programma che permette di calcolare i valori approssimati di $\sqrt{1}$, $\sqrt{2}$, ... $\sqrt{100}$.*

Si ricorda il principio di interpolazione lineare:

sia $x_1 < x < x_2$

Conoscendo i punti: $\begin{cases} x_1 \\ f(x_1) \end{cases}$ $\begin{cases} x_2 \\ f(x_2) \end{cases}$

il valore $f(x)$ è ottenuto prendendo il valore approssimato che corrisponde alla retta che unisce i due punti:

$$\frac{f(x_2) - f(x_1)}{f(x) - f(x_1)} = \frac{x_2 - x_1}{x - x_1}$$

7. *Scrivere un programma che permetta di calcolare la radice quadrata negativa di 0.25, partendo da $x_0 = -0.6$ e applicando il seguente metodo:*

$F(x) = x^2 - 0.25$

$x_n = x^2_{n-1} + x_{n-1} - 0.25$

Si richiede una precisione di almeno 10^{-3}.

8. *Si abbia l'equazione del tipo:*
$F(x) = a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 = 0$
Essa rappresenta una equazione di sesto grado (anche se qualche coefficiente può essere nullo).
$a_0, a_1, \ldots, a_6$ *sono dei dati.*
Scrivere un programma che legga questi dati, calcoli $F(x_0)$*, e stampi i coefficienti,* x_0 *e* $F(x_0)$ *per* x_0 *che varia da* -10 *a* $+10$ *con un passo di 0.1.*

9. *Si consideri la seguente sequenza:*
1 — E prende il valore 1.
2 — D prende il valore 5.
3 — Sostituire E con 1 + (E.X/D).
4 — Se D = 1 STOP. Altrimenti sottrarre 1 da D e tornare al 3.
Quando l'aigoritmo, si arresta,

$$E = 1 + x + \frac{x^2}{2} + \frac{x^3}{2.3} + \frac{x^4}{2.3.4} + \frac{x^5}{2.3.4.5.} = e^x$$

Scrivere il programma supponendo che X venga letto come un dato.

10. *Applicare questo metodo a*

$$\text{cotg}(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9}$$

Scrivere il programma.

11. *Come si potrebbe calcolare* 2^{200} *in maniera esatta (numero elevatissimo).*

12. *Il minimo comune multiplo è il più piccolo intero divisibile per A e B. Scrivere il programma che permetta di calcolare il minimo comune multiplo tra due interi A e B.*

13. *A è una tabella (20, 20). Scrivere un programma che calcoli la somma dei valori assoluti degli elementi della Kesima riga di A, salvo A (K, K).*

$$\text{SOM ABS} = \sum_{j \neq k} |A_{kj}|$$

14. *Dato un campione di N valori, si vuole selezionarne alcuni.*
   a) *Si vogliono selezionare i valori compresi tra* $m - 2\sigma$ *e* $m + 2\sigma$*, sapendo che m rappresenta la media degli n valori e* σ *rappresenta lo scarto tipo calcolato così:*

$$\sigma = \sqrt{\sum_{i=1}^{N} (x_i - m)^2 \, P(X = x_i)}$$

se x_i rappresenta un elemento della tabella, si supporrà in questo caso che:

$$P(X = x_i) = \frac{1}{N}.$$

b) *I valori selezionati in a) sono inseriti in una seconda tabella. Si vuole selezionare ora la serie dei valori tali che:*

$$m - \frac{m_{i-1}}{2} \leqslant x_i \leqslant m + \frac{m_{i-1}}{2}$$

con:

$$m_{i-1} = \frac{1}{i-1} \sum_{j=1}^{i-1} x_j$$

c) *Scrivere lo schema a blocchi e il programma BASIC che facciano le seguenti operazioni:*
1) Leggere una tabella A di 50 posizioni e stamparla.
2) Fare una prima selezione sulla tabella A e trasferire i valori che rispondono ai requisiti richiesti in una tabella B e stamparla.
3) Fare una seconda selezione, partendo dalla tabella B e trasferire i valori risultanti in una tabella C e stamparla.
4) Ordinare i valori di C e stampare la tabella ordinata.

CAPITOLO 5

IL TRATTAMENTO DEGLI ARCHIVI IN BASIC («FILES»)

Nei capitoli precedenti, abbiamo visto le istruzioni standard e le loro varianti, comuni alla maggior parte dei BASIC. In questo capitolo vedremo i trattamenti che sono, per alcuni versi, più elaborati, ma che possono variare notevolmente, a seconda che si disponga di unità di memoria secondarie del tipo cassette magnetiche o dischi flessibili, o comunque di grossi sistemi ad hard disk. Alla stessa maniera, per i trattamenti grafici, tutto ciò dipende dall'unità di visualizzazione di cui si dispone: puramente alfabetici, semigrafici o grafici.

Non è nello scopo di questo libro esaminare tutte le varianti: ci limiteremo dunque a mostrare il tipo di trattamento e cosa è possibile realizzare in materia di trattamento di («files») archivi e di trattamenti grafici, tenendo conto dei tipi di sistemi di cui si dispone.

ELEMENTI DI BASE SUGLI ARCHIVI (FILES) INFORMATICI

1. LA NOZIONE DI ARCHIVIO

In informatica, la nozione di archivio è una nozione generale e, per quanto riguarda il suo contenuto, semplicissima.

Definizione

Un archivio o file è una serie di informazioni binarie (bits) registrate su un supporto di memoria e definite da un identificatore.

In modo più pratico, si può dire che si tratta di una serie di dati numerici o non numerici registrati su un supporto di memoria diverso dalla memoria centrale. In particolare, in questo caso, possiamo parlare di archivi, di archivi

perforati, di nastri perforati, di archivi stampati (essendo la carta stessa un supporto di memoria), ma, più spesso, il supporto utilizzato è di tipo magnetico. Si può anche parlare di schedari «programmi», poichè, per la definizione data, un programma può essere considerato come archivio, quando è immagazzinato in una memoria secondaria.

2. SUPPORTI DI ARCHIVIO

Abbiamo visto che la nozione di supporto di memorizzazione è essenziale per parlare di schedario in senso informatico.

Agli inizi, i supporti principali per files erano le schede perforate e/o i nastri perforati. Questo tipo di supporto, benchè ancora utilizzato, tende a scomparire, in quanto tali archivi sono difficili da manipolare, da immagazzinare e lunghi da trattare a causa della lentezza del lettore di schede o di nastri.

In uscita, abbiamo visto che i listing possono essere considerati come files e, benchè si tratti di un tipo di supporto certamente necessario, si tratta semplicemente di archivi che possono essere scritti dalla macchina.

Continuando il capitolo, ci interesseremo quindi principalmente dei files su supporto magnetico, che hanno il vantaggio di poter essere letti e scritti dalla macchina a velocità relativamente elevata.

Esse sono poi meno ingombranti e più facilmente manipolabili per trattamenti informatici. Il loro solo difetto è che è necessario avere un calcolatore per trattarle, sia in lettura che in scrittura.

Tra questi supporti si distinguono: i supporti di tipo *sequenziale* (bande magnetiche o cassette magnetiche) e i supporti di tipo *aleatorio*, come i dischi magnetici.

2-1. I supporti di tipo sequenziale

Nei medi e grossi sistemi (ivi compresi i mini-elaboratori), si dispone di *bande magnetiche* informatiche dove la registrazione è digitale (binaria) e i codici sono normalizzati seguendo dei precisi standard. Questo tipo di banda magnetica permette di registrare fino a diversi milioni di caratteri con densità di registrazione che attualmente varia da 800 a 9.600 BPI (Bit Per Inch). Questo tipo di supporto è dunque interessante per immagazzinare dei grossi schedari, in particolare per schedari archivi, ma non è disponibile su sistemi di microelaboratori, perché il suo costo è proibitivo.

Di contro, esistono supporti sequenziali di tipo *cassette magnetiche.*

Bisogna sottolineare che esistono tuttavia due tipi di sistemi di cassette: le cassette digitali e le cassette audio.

Nelle *cassette digitali*, l'informazione è registrata seguendo lo stesso principio valido per le bande magnetiche standard.

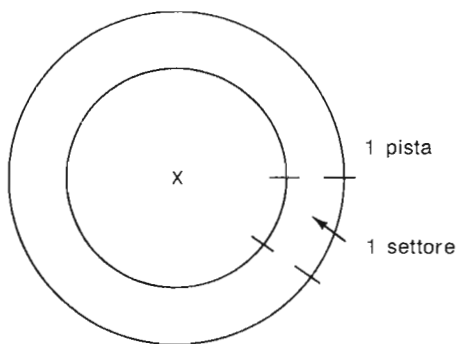
Per ciò, esse hanno il pregio di essere più sicure delle bande magnetiche, possono essere lette assai rapidamente, per contro, i registratori di tali cassette sono relativamente costosi. Il loro uso è considerabilmente diminuito con l'apparizione dei floppy disck (dischi flessibili) e l'utilizzazione dei sistemi di cassette audio. Il tipo di supporto più economico e più esteso sui piccoli sistemi microelaboratori è attualmente la *cassetta audio*, utilizzata con dei normali registratori di cassette. Bisogna notare tuttavia che non esiste nessuno standard al livello della codificazione delle informazioni e che, a seconda dei sistemi utilizzati, l'affidabilità è più o meno grande. D'altra parte, la lettura dei dati su tali sistemi è un processo assai lungo, tenuto conto delle velocità ridotte sia in lettura che in scrittura.

Per dei piccoli schedari, trattati in maniera sequenziale, e specialmente per l'immagazzinamento di archivi programma, questa soluzione è ancora accettabile.

Nel seguito del capitolo, daremo alcuni esempi di trattamento di schedari di questo tipo.

2-2. I supporti di tipo aleatorio

In contrapposizione al supporto sequenziale, un supporto è detto aleatorio, se si può accedere all'informazione in maniera diretta o aleatoria, senza dover far scorrere l'insieme del supporto dall'inizio alla fine, per leggere o scrivere un dato. Ciò non significa che il supporto si comporta in maniera aleatoria in senso probabilistico: questi è sempre comandato dalla macchina in senso deterministico; ciò che è aleatorio è il tempo in cui una informazione verrà letta o scritta, tenendo conto delle letture o delle scritture effettuate precedentemente.

Tra questi tipi di supporti si distinguono gli hard disks e i floppy disks. In tutti i casi, questi dischi sono costituiti da piste concentriche.

Una pista è divisa in settori. Ciascuna pista e ciascun settore sono contraddistinti da un numero che permette di specificare la lettura o la scrittura di tale pista o tale settore. Questo numero si chiama *indirizzo*.

a) *Gli hard disks*

I dischi di questo tipo sono caratterizzati da un supporto rigido ricoperto da una sostanza magnetica e chiuso in una cartuccia o in una cassetta.

In questo caso, le testine di lettura o di scrittura non si appoggiano sul supporto magnetico, e ciò permette di ottenere delle velocità abbastanza notevoli senza provocare l'usura del supporto.

In questa categoria si distinguono i *dischi a testine fisse*, in cui esistono sia le testine di lettura che le piste. Il supporto ruota e le testine sono fisse. Ciò permette di accedere più rapidamente a ciascun settore. Il tempo necessario per fare questa operazione si chiama *tempo di accesso*. Per questo tipo di dischi, è dell'ordine di 10 millisecondi in media. Questa categoria di dischi è la più costosa e presenta l'inconveniente che non si può rimuovere. A causa di ciò, non possono essere usati per schedari d'archivi. Sono infatti utilizzati solo su quei sistemi in cui il tempo d'accesso è un parametro cruciale per il funzionamento del sistema stesso.

La seconda categoria è la categoria dei *dischi a testine mobili*, in cui si ha una sola testina di lettura che può disporsi sulla pista desiderata. Ciò implica il movimento di un braccio che allunga il tempo di accesso (che può essere dell'ordine da 20 a 100 millisecondi, in media 60 ms). Questi dischi presentano il vantaggio di essere meno costosi e di essere amovibili, cosa che permette a più supporti di essere montati l'un dopo l'altro e di permettere l'archiviazione. La loro capacità varia da più milioni di caratteri a più decine di milioni di caratteri, nel caso in cui più supporti sono accatastati l'uno sull'altro («dispaks»). In tutti i casi, tuttavia, si tratta di unità di registrazione relativamente costose, anche se gli sviluppi della tecnologia permettono di prevedere grosse riduzioni di costo (dischi tipo «Winchester») per i dischi non amovibili.

Questo tipo di dischi è necessario se si vuol trattare un grande numero di dati e specialmente per i sistemi di base dei dati. Sui grossi e medi sistemi, ivi compresi i sistemi a minielaboratori, questo tipo di dischi costituisce ciò che si chiama «memoria di massa», che può essere utilizzata per più applicazioni e da più utenti nello stesso momento.

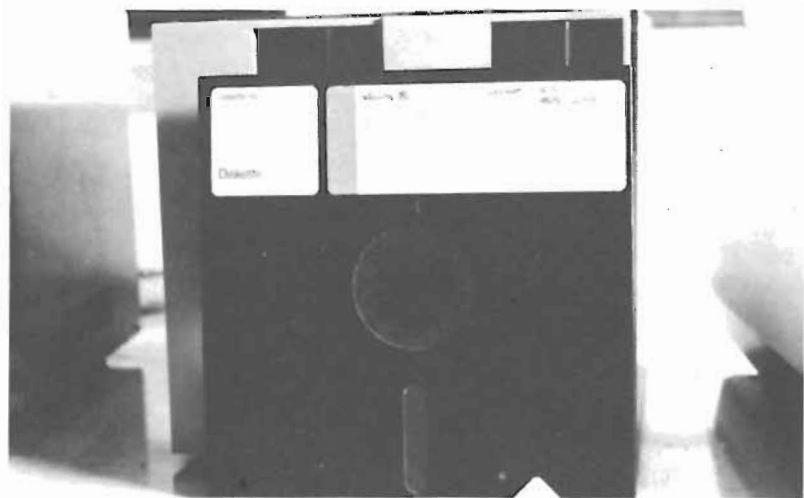
b) *I dischi flessibili*

Il principio fisico di registrazione e di accesso alle informazioni è lo stesso visto per i dischi rigidi.

Tuttavia, differiscono per il materiale di supporto (plastica ricoperta da substrato magnetico) e per la meccanica, molto più semplice dei registratori.

Si tratta di dischi a testine mobili, ma, alcune volte, la testina di lettura è appoggiata su un supporto.

I pregi di questi dischi sono il loro basso costo e le loro dimensioni ridotte.

Ne esistono differenti modelli, dei quali il solo standard attuale è lo standard IBM per i dischi con una capacità di circa 250.000 caratteri.

disco flessibile

Con prestazioni minori si hanno poi i minidischi che contengono circa 100000 caratteri.

Da qualche anno si trovano in commercio i dischi «doppia faccia-doppia densità» che permettono di quadruplicare le capacità dei modelli esistenti. Su un disco è possibile dunque immagazzinare più di un milione di caratteri. Su un minidisco si può arrivare fino a 500000 caratteri.

Tuttavia, bisogna tener presente che, per i minidischi, non esiste alcun standard; questo comporta, per i differenti sistemi, dei problemi di lettura dei dischi.

Lo sviluppo di questi dischi permette poi di avere delle piccole memorie di massa associate al microelaboratore.

L'affidabilità di questi supporti attualmente è molto buona ed, escludendo l'usura, a lungo termine essi costituiscono indubbiamente il sistema più economico per immagazzinare in modo diretto o aleatorio.

3. I METODI DI ACCESSO AGLI ARCHIVI («FILES»)

Per metodo di accesso si intende il modo in cui un archivio può essere trattato da un programma, indipendentemente dal supporto in cui si trova.

3-1. Archivi ad accesso sequenziale

In questo tipo di archivi le informazioni sono registrate sequenzialmente le une dopo le altre. Quando si vuole leggere un dato appartenente a questo archivio, bisogna dunque leggere tutti i dati che lo precedono. Allo stesso modo se si vuole scrivere o aggiungere un dato nell'archivio, bisogna posizionarlo alla fine. Questo tipo di accesso è il più semplice da mettere in opera e da trattare, ma è tuttavia il meno flessibile. Per contro, è proprio adatto ai supporti sequenziali come i nastri o le cassette magnetiche.

Bisogna tuttavia notare che si possono avere archivi ad accesso sequenziale su supporti che non lo sono.

In questo capitolo vedremo, in seguito, degli esempi di trattamento di archivi sequenziali.

Tali archivi sono in genere organizzati in registri successivi di grandezza fissa o variabile, a seconda dei programmi che li trattano.

3-2. Archivi ad accesso diretto

In questo tipo di archivio, basta precisare il numero o una chiave che permette di identificare il dato o il registro in cui si trova questo dato per averne direttamente accesso, indipendentemente dalla posizione fisica nell'archivio considerato. Questa tecnica può essere realizzata in modo pratico solamente su supporti di tipo aleatorio. In alcuni casi, questo metodo di accesso si chiama aleatorio, in quanto si passa attraverso l'intermediazione di una funzione di ripartizione aleatoria delle informazioni sul disco, funzione che ha per parametro la chiave associata a questi dati. Questo tipo di accesso è assai vantaggioso quando si voglia un accesso rapido ai dati.

3-3. Archivi ad accesso indicizzato

Si tratta di un metodo di accesso diretto o «semidiretto» di tipo particolare, poiché si deve, prima di tutto, ricercare sequenzialmente il valore della chiave in una tabella chiamata indice e, quando si è trovata questa chiave, si ottiene il posto esatto dei dati o del registro associato.

Questo tipo di accesso presenta il vantaggio di dare un accesso rapido, se gli indici non sono troppo grandi; conservando comunque la possibilità di trattare gli archivi sequenzialmente, controllando gli indici, secondo un ordine definito.

Questo tipo di accesso sarà dunque utilizzato quando si voglia trattare archivi in maniera sequenziale, avendo la possibilità di accedervi in modo rapido e quasi diretto.

3-4. Il metodo detto sequenziale indicizzato

Questo metodo è, per quanto riguarda il principio, simile al precedente, ma corrisponde ad indici che sono definiti e amministrati dal sistema di gestione della macchina. In questo caso, l'indice corrisponde non ad un valore di chiave isolato, ma ad un gruppo di chiavi di cui l'indice precisa il primo e l'ultimo elemento.

Successivamente, si ha la ricerca del registro desiderato in un sottoinsieme di archivi, e questa ricerca viene fatta sequenzialmente. Con questo metodo, si possono esaminare egualmente più livelli di indici.

Questa tecnica è dunque relativamente complessa come esecuzione e si mostra interessante soltanto quando il numero di chiavi possibili è estremamente elevato. Tuttavia, bisogna rilevare che questo metodo di accesso è molto utilizzato in gestione e che comincia ad essere utilizzato su sistemi di microelaboratori orientati verso applicazioni di gestione.

3-5. L'organizzazione degli archivi

Qualunque sia il metodo di accesso, gli archivi sono, in generale, organizzati in blocchi od entità, chiamati *registri logici.* Ciò significa che la dimensione dell'archivio è indipendente dalla struttura fisica dei registri sul supporto considerato.

Dal punto di vista di chi programma, l'unica parte interessante è il registro logico. La dimensione di questi registri può essere fissa o variabile a seconda delle applicazioni. La disposizione di questi sul supporto può essere fissa o meno.

Nel caso che sia fissa, la si associa, per esempio, ad un settore a più registri, in modo tale da permettere un miglior uso dello spazio disponibile.

Quando può essere trattato sequenzialmente, uno schedario termina sempre con un registro particolare detto *fine di archivio* («end of file»). Il rilevare questa fine di archivio permette dunque di sapere che il trattamento è finito.

4. I SISTEMI DI GESTIONE DEI DISCHI

Vengono chiamati in gergo informatico i DOS («Disk Operating System»). Sono costituiti da un certo numero di programmi sviluppati dal costruttore e facenti parte del sistema logico fornito all'utente per la gestione di schedari su dischi. Un tale sistema comprenderà, dunque, almeno una gestione di tavole di identificazione di archivi e di spazio disponibile sul disco.

Sui grossi e medi sistemi, il DOS può essere estremamente complesso e può permettere la gestione di archivi sequenziali indicizzati o diretti.

Sui sistemi micrordinatori, le possibilità sono più ridotte, ma stanno rapidamente evolvendosi, specialmente sui sistemi destinati alle applicazioni di gestione.

Senza arrivare a descrivere i differenti tipi di DOS che si possono incontrare, basta sapere che, per sviluppare un'applicazione di trattamento di archivi, bisogna disporre di un minimo di applicazioni che si chiamano le primitive del sistema di archivi e che sono a disposizione dell'utente.

Le primitive più usuali sono: la creazione di un archivio, la sua apertura, la lettura e la scrittura di un registro o dei dati elementari, la chiusura di un archivio, o la sua distruzione.

Il DOS comprende in generale un certo numero di programmi del tipo: la copia di archivi, la verifica delle tavole degli archivi, gli elenchi dei nomi degli archivi scritti su un supporto, la visualizzazione del contenuto di un archivio...

Alcune di queste operazioni possono certamente essere programmate dall'utente, se dispone di funzioni elementari di un sistema di gestione di archivi.

Per quanto concerne il trattamento di archivi nel BASIC, è certamente necessario disporre di queste primitive, che si tradurranno mediante istruzioni particolari.

La seconda parte di questo capitolo riguarda lo studio di queste istruzioni e la loro programmazione, eseguita utilizzando semplici esempi.

5. IL TRATTAMENTO DEGLI ARCHIVI SU SUPPORTO SEQUENZIALE IN BASIC

Le primitive del trattamento di archivi sequenziali

Si supponga, per esempio, di disporre di un supporto sequenziale di tipo cassetta magnetica.

Un consiglio per gli utenti che vogliono verificare programmi comportanti la scrittura di dati negli archivi: utilizzare sempre dei supporti vergini per evitare il non augurabile «schiacciamento» degli archivi o dei programmi. D'altra parte, prima di iniziare ad immagazzinare un archivio reale, bisogna assicurarsi che il programma sia corretto.

Per mezzo di questa raccomandazione, bisogna innanzi tutto presentare le istruzioni di base necessarie per la programmazione di istruzione per il trattamento di archivi sequenziali.

a) *apertura di un archivio*

Questa operazione serve a preparare l'archivio e a definire un certo nume-

ro di parametri che permetteranno di sapere se si tratta di una lettura o di una scrittura o di precisare su quale unità deve trovarsi l'archivio quando esistono più registratori. L'operazione permette anche di precisare il nome dell'archivio.

La sintassi di questa istruzione può variare a seconda del tipo di macchina. Noi utilizzeremo l'esempio di un microelaboratore «PET» della Commodore.

In questo caso si ha un'istruzione del tipo:

APRI n° logico di archivio, n° dell'unità, codice di lettura-scrittura, nome dell'archivio

- *Il numero logico* di archivio è un numero particolare nel programma che servirà a individuare questo archivio in tutto il programma. Il numero può variare da 1 a 255, e può aprire simultaneamente fino a 10 archivi.

- *Il numero dell'unità* indica il numero fisico dell'inserimento in registro utilizzato, nel caso se ne abbiano a disposizione più di uno. Se il numero non è indicato, il valore per difetto è 1 e corrisponde al registratore incorporato nella macchina. Questo numero può teoricamente andare da 1 a 255, ma se ci si rivolge a una unità non esistente, ciò provoca un errore.

- *Il codice di lettura-scrittura:* se si precisa 0 è una lettura, se 1 è una scrittura, se 2 è una scrittura con la stampa di un simbolo «fine di banda» alla chiusura dell'archivio.

- *Il nome dell'archivio:* è una serie di caratteri che permettono di identificare l'archivio stesso. Non mettendo niente, il nome verrà considerato come «bianco».

Il nome può essere lungo fino a 80 caratteri.

Esempi:

- OPEN 1, 1, 1, "DOSSIER"
  permette di aprire l'archivio n° 1 del programma sul registratore di cassetta n° 1 in scrittura, su un archivio chiamato «DOSSIER».
- OPEN 1, 1, 0, "DOSSIER"
  permette di aprire lo stesso archivio in lettura.

b) *chiusura di un archivio*

Questa istruzione permette di indicare che le operazioni di lettura o scrittura su un archivio sono terminate. Nel caso in cui lo schedario era stato aperto in scrittura con l'opzione 2, viene scritto un simbolo «fine di banda».

All'inizio, tutti gli archivi aperti in un programma devono essere terminati prima della fine del programma, altrimenti si rischia di ottenere degli errori durante la rilettura di questi archivi. L'istruzione non precisa che un solo parametro, che rappresenta il numero logico dello schedario.

CHIUDI n° logico di archivio
(CLOSE)

Esempio: CLOSE 1
chiudi l'archivio n° 1

c) *Scrittura di dati in un archivio*

Una volta aperto l'archivio in lettura, è sufficiente dare un ordine in uscita sull'archivio corrispondente, precisando l'elenco delle variabili contenente i dati da scrivere.

Il principio è dunque analogo a quello di un ordine PRINT e l'istruzione è in effetti identica con in più la necessità di specificare il numero dell'archivio logico in questione.

La sintassi di questa istruzione è:
STAMPA # n° logico, elenco delle variabili
(PRINT)

Quando i nomi delle variabili dell'elenco sono separati da virgole, i dati saranno registrati nell'ordine precisato e saranno seguiti da ®, il cui codice in ASCII è 13.

Quando i nomi delle variabili sono separati da punto e virgola, i valori delle variabili sono registrati senza il separatore ®. Tuttavia, se si vogliono rileggere dei dati con un ordine ENTER, bisogna inserire un separatore ® dopo il 79° carattere (vedi in seguito).

d) *La lettura dei dati di un archivio*

Si può fare secondo due tecniche che abbiamo già visto per l'ingresso da tastiera. La prima tecnica concerne la lettura di un dato nel suo insieme, sia il dato numerico o no. Per far ciò, si utilizza l'ordine INPUT.

La struttura di questa istruzione è:
ENTRA # n° logico, elenco delle variabili
(PRINT)

Ciò permette di leggere i dati come se essi venissero dalla tastiera, ma partendo da un archivio. Le variabili possono essere numeriche o stringhe di caratteri. In particolare, ciò implica che non si possono avere sequenze di caratteri di lunghezza superiore a 79 tra due caratteri ®.

Questa limitazione è in funzione della macchina utilizzata e potrebbe essere facilmente soppressa su altre versioni.

L'altro modo di leggere i dati è di leggerli carattere per carattere: per fare ciò, viene utilizzata l'istruzione GET.

Di questo modo esistono due forme, a seconda che si vogliano leggere delle numeriche o dei caratteri.

La prima forma è la seguente:
GET # n° archivio logico, variabile numerica.

In questo caso, se si ha un carattere da 0 a 9, la variabile contiene il valore corrispondente. Se si tratta di un carattere + — o bianco, la variabile contiene il valore 0 o — 1.

Tutti gli altri caratteri daranno un errore (ivi compreso il simbolo di fine di archivio).

La seconda forma è più generale e utilizza una variabile stringa di caratteri.
GET # n° di archivio logico, variabile stringa.

In questo caso la variabile contiene il carattere letto, e tutti i caratteri sono accettati.

Questa istruzione permette in particolare di leggere dati con più di 79 caratteri senza separatore ®.

Il solo problema è di sapere quando fermarsi. Per fare ciò, bisogna poter verificare la fine dell'archivio o la fine della banda. Allo stesso modo, bisogna poter verificare se si sono avuti errori di lettura.

Ciò è effettuato generalmente attraverso un test di un codice di stato.

e) *Utilizzazione di una parola di stato*

La parola di stato è una parola memoria (Byte) che contiene un codice indicante il risultato dell'operazione di entrata uscita su un archivio.

Esempio

Se si vuole verificare la fine di un archivio, si potrà utilizzare una istruzione.

SE	(ST)	E	64	ALLORA
IF	(ST)	AND	64	THEN

Bisogna allora considerare (ST) E 64 come una espressione logica vera se il bit 6 è posizionato in ST ($2^6 = 64$).

Per contro, volendo accertarsi dell'assenza di errori, è sufficiente utilizzare una istruzione

IF ST < > 0 THEN

Nota. — Questo tipo di funzionamento non è standardizzato, ma è sicuramente utile, se si utilizzano cassette magnetiche la cui affidabilità non è a tutta prova.

Questa parola di stato, chiamata ST (status) sul «PET», può assumere differenti valori, a seconda delle seguenti condizioni:

- *blocchi troppo corti:* se si leggono dei dati e si incontra un delimitatore prima del numero atteso di caratteri (il codice è 4);
- *blocchi troppo lunghi:* se si leggono dei dati e non si incontrano delimitatori, quando il numero dei caratteri atteso è stato letto, il codice è allora 8;
- *errore fatale di lettura:* è impossibile leggere un dato; il codice è allora 16;
- *errore di controllo:* i dati sono stati letti, ma il controllo effettuato sull'insieme dei dati non concorda con la parola di controllo letta sul supporto (codice 32); in questo caso si può riprovare la lettura;
- *fine dell'archivio* (codice 64): è posizionato quando la fine dell'archivio è incontrata durante la lettura;
- *fine della banda* (codice 128): ciò permette di sapere che aldilà non c'è più archivio.

Questa parola di stato può essere verificata da un programma, grazie all'uso di operatori logici AND, OR, NOT.

Un'operazione logica AND, in effetti, legata alla parola di stato, permette di sapere se un bit è posizionato o no.

Esempio:

Si abbia un programma che richiama delle parole e le scrive in un archivio ARCH. Ci si arresta quando si incontra la parola FINE.

Successivamente si ribobina la cassetta e si fa l'operazione inversa di lettura.

Ciò è dato dal seguente programma:

```
10     OPEN 1, 1, 2, "ARCH"
20     INPUT Q$
30     PRINT #1, Q$
40     IF Q$ < > "FINE" THEN 20
50     CLOSE 1
60     INPUT "RIBOBINARE LA CASSETTA"; N
70     OPEN 1, 1, 0, "ARCH"
80     INPUT #1, Q$
90     PRINT Q$;
100    IF Q$ < > "FINE" THEN 80
110    CLOSE 1
120    END
```

Esecuzione:

```
?   NOI ®
?   SCRIVIAMO ®
?   DELLE ®
?   PAROLE ®
?   SULLO ®
?   ARCHIVIO ARCH ®
?   FINE
RIBOBINARE LA CASSETTA! 1 ®
NOI SCRIVIAMO DELLE PAROLE
SULL'ARCHIVIO ARCH FINE
```

6. IL TRATTAMENTO DEGLI ARCHIVI SU DISCHI

Abbiamo visto che ciò necessita di un supporto logico del tipo «DOS», che si può chiamare il sistema di gestione di archivi su disco (SGAD).

Questo SGAD è accessibile, per un verso, con l'aiuto di comandi sistema che fanno parte del monitor, e, per un altro, con l'aiuto di primitive di gestione di archivi che possono essere utilizzati in un programma.

Così, per esempio, se si vuole aver accesso ad un elenco di archivi su disco, si utilizzerà un comando che chiede quest'elenco. Può essere un comando del tipo:

ANNUARIO ® (DIRECTORY)
o CATALOGO ®

Per contro, se si vuole scrivere su un archivio, si utilizzerà una istruzione di entrata - uscita.

Presenteremo prima i comandi, poi le istruzioni.

Prendiamo il caso di un sistema di archivio su dischi flessibili, gli esempi sono presi su un sistema APPLE con doppia unità di dischi. Si intende che comandi analoghi devono esistere su tutti i sistemi aventi un DOS.

6-1. Inizializzazione di un dischetto

Se si dispone di un dischetto vergine (appena fornito dal costruttore), bisogna preliminarmente realizzare una operazione detta di inizializzazione del supporto. In effetti, ogni sistema dispone di una tecnica propria per organizzare i dati su disco. Questa operazione è necessaria anche se, per esempio, si utilizza uno standard IBM.

Questa istruzione può avere più possibilità.

Sul sistema APPLE, per esempio, si avrà:

INIT NOME ARCH, n° del lettore registratore, n° del volume, n° di controllo.

Il nome dell'archivio può essere scelto liberamente, e serve semplicemente all'archivio che permette di immagazzinare un programma di messa in moto del SGAD (è il «Bootstrap»).

Gli altri parametri sono:

- il numero dell'unità del disco utilizzato, nel caso siano più di uno (il parametro incomincia per D come «Drive»).
- il numero di volume che è il numero assegnato al disco (da 1 a 254). Questo parametro incomincia per V;
- il numero di controllo di entrata-uscita (varia da 1 a 7). Se è omesso, si prende il valore 1 per difetto (lettera S come «Slot»).

Esempio:

INIT CIAO, D1, V2, S1

inizializza un disco con un archivio iniziale chiamato "CIAO" sull'unità n° 1, con un numero di volume 2 e con il controllo n° 1.

6-2. Catalogo di un dischetto

L'elenco degli archivi disponibili sul disco può essere ottenuto grazie a un comando:

CATALOG, D n° dell'unità

se il numero è assente, il valore per difetto è messo a 1.

Esempio:

CATALOG, D2

visualizza o stampa il catalogo della seconda unità di disco.

Nello stesso tempo, si può aver visualizzazione del tipo di archivio e del posto che occupa sul disco (numero di settore).

6-3. Immagazzinamento degli archivi programmi

Una delle prime utilità di un SGAD è di permettere l'immagazzinamento dei programmi su disco.

Esiste un comando che permette di fare ciò, vale a dire di salvaguardare il programma attualmente in memoria centrale su disco.

Si tratta del comando:

SAVE NOME DEL PROGRAMMA, Dn

Si precisa il nome del programma che sarà allora considerato come un nome di archivio su disco.

Dn è il parametro numero di unità.

Esempio:

SAVE VIE, D2

registrerà il programma attualmente in memoria sotto il nome di archivio VIE sul disco n° 2.

Attenzione: bisogna fare attenzione agli omonimi! Se un archivio non è protetto in scrittura, può essere ricoperto da un altro archivio con lo stesso nome.

6-4. Lettura in memoria di un archivio programma

È l'operazione inversa e consiste nel leggere un archivio programma situato su disco.

Ciò si chiama caricamento del programma in memoria.

Si tratta del comando:

LOAD NOME DELL'ARCHIVIO, Dn
(CARICARE)

Il significato dei parametri è lo stesso che nella precedente operazione.

Se il nome non esiste nel catalogo, viene inviato un messaggio d'errore.

6-5. Caricamento con lancio di un programma

Si tratta di una operazione identica alla precedente, ma con domanda di esecuzione del programma.

Il comando da utilizzare è

RUN NOME ARCHIVIO PROGRAMMA, Dn
(CAMMINA)

6-6. Cancellazione di un archivio

Volendo distruggere un archivio esistente, si utilizza un comando di cancellazione:

DELETE NOME ARCHIVIO, Dn
(CANCELLA)

Questo comando cancella il nome dell'archivio nel catalogo e libera il posto occupato dall'archivio stesso, a condizione che questi non sia protetto.

Questo comando è valido, qualunque sia il tipo di archivio.

6-7. Protezione e deprotezione degli archivi

Per evitare le distruzioni accidentali o intempestive degli archivi è prudente proteggerli. Questa operazione si attua con il comando:

LOCK NOME ARCHIVIO, Dn
(CHIUDI A CHIAVE)

Così l'operazione di riscrittura o di cancellazione di questo archivio sarà interdetta.

Se, al contrario, si vuole realizzare l'operazione inversa, si utilizzerà il comando:

UNLOCK NOME ARCHIVIO, Dn

L'archivio ridiventa allora «vulnerabile».

Nota. — Volendo proteggere il contenuto di un disco completo, sarà meglio far ricorso a un meccanismo di protezione scrittura fisica, che viene realizzata grazie a un anello di protezione sull'involucro del disco.

Inversamente, se si vuole togliere la protezione ad un disco, si può farlo togliendo l'anello.

6-8. Cambiamento del nome di un archivio

In alcuni casi, si può aver bisogno di dare un altro nome ad un archivio. Questa operazione viene attuata mediante il comando:

RENAME ANTICO NOME, NUOVO NOME, Dn
(RINOMINA)

Questa operazione mette in luce il catalogo, sostituendo l'antico nome con il nuovo, senza modificare il contenuto dell'archivio.

6-9. Utilizzo dei comandi in un programma BASIC

Tutti questi comandi sono ugualmente accessibili con l'intermediazione dell'interprete BASIC sul sistema APPLE.

Per fare ciò, bisogna utilizzare delle istruzioni di uscita (PRINT), il primo carattere delle quali è un carattere di controllo (CONTROLLO D o D^c). Alla tastiera, questo carattere è ottenuto premendo simultaneamente i tasti CONTROL e D. Il codice ASCII di questo carattere è il numero 4.

Il carattere D^c può essere definito da:

D$ = CHR$ (4)

cioè il carattere avente per codice ASCII: il numero 4. Questo carattere deve essere seguito dalla catena di caratteri corrispondente al comando che si vuole effettuare.

Esempio:

```
10      D$ = CHR$ (4)
20      PRINT D$; "CATALOGO"
```

ha come effetto la stampa o la visualizzazione del CATALOGO.

IL TRATTAMENTO DEGLI ARCHIVI SEQUENZIALI SU DISCO

Consideriamo ora archivi di dati trattati da programmi BASIC.

Questi trattamenti si richiamano al SGAD (DOS) con l'intermediazione delle primitive o istruzioni al SGAD.

Queste primitive non sono accessibili al di fuori di un programma. Qualunque sia il sistema, le funzioni di queste primitive sono sempre le stesse:

APERTURA di un archivio (OPEN)
CHIUSURA (CLOSE)

LETTURA	(READ)
SCRITTURA	(WRITE)
AGGIUNTA	(APPEND)
POSIZIONAMENTO	(POSITION)

Per definire queste primitive, si può scegliere tra l'uso di parole chiave (OPEN, READ, WRITE, ...) e l'introduzione di nuove istruzioni BASIC o, dove è possibile, utilizzare le istruzioni di entrata-uscita standard, distinguendo le primitive al SGAD con un carattere speciale.

Nella maggior parte dei BASIC, è stata scelta la prima soluzione, ma non esiste nessuna standardizzazione. La seconda soluzione è utilizzata sul sistema APPLE: presenta il vantaggio di definire programmi sintatticamente standardizzati. Noi utilizzeremo questa soluzione.

Definizione di una primitiva

Il metodo utilizzato è identico a quello già visto per la definizione dei comandi al SGAD in BASIC.

Il carattere speciale utilizzato è il carattere CONTROLE D (D^c) che ha per codice ASCII il 4.

Generalmente, avendo definito D$ = CHR$ (4), una primitiva al SGAD è definita con l'istruzione:

PRINT D$; "PRIMITIVA"

Il testo della primitiva è espresso tra virgolette.

Si può anche definire una variabile PR$, tale che

PR$ = "TESTO DELLA PRIMITIVA"

quindi l'istruzione PRINT D$; PR$.

Struttura di un archivio sequenziale in BASIC

In BASIC, un archivio o «file» è una serie di caratteri (codificati in ASCII). Questi caratteri sono organizzati in articoli, campi o items, separati da caratteri ® e aventi come codice ASCII il 13.

Esempio:

Si abbia un archivio contenente indirizzi di persone. In questo archivio si ha, per esempio, la serie di caratteri:

BIANCHI MARIO ® VIA MAGENTA, 42 ® 20100 ® MILANO ®

Questa serie di caratteri è registrata sequenzialmente nell'archivio, essendo ciascun carattere codificato in ASCII. Il carattere ® serve da separatore tra i vari items.

LE DIFFERENTI PRIMITIVE DI UN SGAD SEQUENZIALE

L'apertura di un archivio

La primitiva corrispondente è
APRI $ = "OPEN NOME ARCHIVIO, Dn, Vm, Sp"
(APRIRE)

L'istruzione è allora:
PRINT D$; APRI$

Questa istruzione ha come effetto di posizionarsi all'inizio dell'archivio in cui il nome è specificato e che si trova sul disco Dn, il volume Vm e associato al controllo Sp.

Nello stesso tempo, il sistema prevede una zona di lavoro in memoria centrale, per effettuare delle letture-scritture dell'archivio.

Se l'archivio non esiste, si ha un messaggio di errore.

Se è già aperto, l'istruzione lo chiude e poi lo. riapre.

Nota. — Il numero di archivi che si possono aprire simultaneamente è limitato ed è precisato dal comando MAXFILES, che permette di avere fino a 16 archivi aperti nello stesso tempo. L'opzione per difetto è di tre archivi simultanei.

Esempio:

```
10        D$ = CHR$ (4)
20        PRINT D$; "OPEN INDIRIZZO"
```

realizza l'apertura dell'archivio INDIRIZZO. Le opzioni per difetto sono D1 e V1.

Chiusura di un archivio

Questa operazione è l'inverso della precedente, in quanto libera lo spazio memoria utilizzato durante l'apertura. D'altra parte, se l'archivio è stato utilizzato in scrittura, ciò permette di scrivere la fine dell'archivio stesso. La dimenticanza della chiusura di un archivio può portare ad una perdita di dati.

La sintassi è:
CLOSE NOME ARCHIVIO
(CHIUDI)

Non è necessario precisare altri parametri, in quanto l'archivio è preliminarmente aperto.

Il comando CLOSE senza nome di archivio permette di chiudere tutti gli archivi.

Esempio:

```
100        PRINT D$; "CLOSE INDIRIZZO"
```

permette di chiudere l'archivio INDIRIZZO.

Scrittura in un archivio sequenziale

Il principio di scrittura è lo stesso che in un archivio su cassetta. Si scrivono degli items separati da caratteri ®. La scrittura presuppone che l'archivio sia stato aperto.

La primitiva che permette di inizializzare una scrittura è:

```
WRITE   NOME ARCHIVIO
(SCRIVI)
```

ma questa primitiva non ha il compito di scrivere nell'archivio, essa indica solamente che tutte le istruzioni di uscita successive a questa istruzione nel corso del programma saranno fatte nell'archivio.

Esempio:

```
 1    REM SCRITTURA DI INDIRIZZI
 5    DIM N$ (100), IN$ (100)
10    D$ = CHR$ (4)
20    PRINT D$; "OPEN INDIRIZZO"
30    SCRIVI$ = "WRITE INDIRIZZO"
35    REM
36    REM INSERIMENTO DEGLI INDIRIZZI
37    REM
40    FOR I = 1 TO 100
50    INPUT "NOME COGNOME"; N$ (I)
55    IF N$ (I) = "ZZ" THEN 80
60    INPUT "INDIRIZZO"; IN$ (I)
70    NEXT I
74    REM
75    REM SCRITTURA SU DISCO
76    REM
```

```
 80     PRINT D$; SCRIVI$
 90     FOR J = 1 TO I
100     PRINT N$ (J)
105     PRINT IN$ (J)
110     NEXT J
120     PRINT D$; "CHIUDI INDIRIZZO"
130     END
```

Esecuzione

```
]RUN
NOME COGNOME GIUSEPPE GARIBALDI
INDIRIZZO CAPRERA
NOME COGNOME UGO FOSCOLO
INDIRIZZO ZANTE
NOME COGNOME ALESSANDRO MANZONI
INDIRIZZO MILANO
NOME COGNOME CARLO LEVI
INDIRIZZO EBOLI
NOME COGNOME ENRICO FERMI
INDIRIZZO ROMA
NOME COGNOME ZZ
```

Questo programma permette di inserire attraverso la tastiera degli indirizzi sotto la forma di nome e cognome, seguiti dalla città, il tutto viene scritto sull'archivio indirizzo.

Incontrando il nome ZZ, si arresta la lettura dei dati, dopodichè il tutto viene scritto nell'archivio.

Lettura di un archivio sequenziale

La lettura presuppone che l'archivio sia stato aperto. La primitiva della lettura indica che tutte le istruzioni di entrata (INPUT) successive vengono effettuate a partire dall'archivio designato dalla primitiva di lettura la cui sintassi è:

```
READ   NOME ARCHIVIO
(LEGGI)
```

Esempio:

Si debba leggere l'archivio indirizzo scritto precedentemente.

```
  1   REM LETTURA DI INDIRIZZI
  5   DIM N$ (100), IN$ (100)
 10   D$ = CHR$ (4)
 20   PRINT D$; "OPEN INDIRIZZO"
 30   LEGGI$ = "READ INDIRIZZO"
 34   PRINT D$; LEGGI$
 35   REM
 36   REM LETTURA SU DISCO
 37   REM
 40   FOR I = 1 TO 100
 50   INPUT N$ (I)
 55   IF N$ (I) = "ZZ" THEN 80
 60   INPUT IN$ (I)
 70   NEXT I
 74   REM
 75   REM STAMPA DEGLI INDIRIZZI
 76   REM
 80   I = I + 1
 90   FOR J = 1 TO I
100   PRINT N$ (J)
105   PRINT IN$ (J)
106   PRINT
110   NEXT J
120   PRINT D$; "CLOSE INDIRIZZO"
130   END
```

Esecuzione

```
]RUN

GIUSEPPE GARIBALDI
CAPRERA

UGO FOSCOLO
ZANTE

ALESSANDRO MANZONI
MILANO

CARLO LEVI
EBOLI
```

ENRICO FERMI
ROMA

In questa maniera, la fine dell'archivio è attuata per mezzo del nome "ZZ". Tuttavia, ciò non è molto pratico se si vogliono aggiungere dei nomi all'archivio stesso. Più in là si vedrà l'opportunità di verificare la fine dell'archivio in modo fisico.

Aggiunta in un archivio sequenziale

Questa operazione consiste nello scrivere nuovi dati alla fine di un archivio già esistente. Per fare questa operazione, si utilizza il comando APPEND (AGGIUNGI).

In questo modo, questa primitiva non scrive realmente, ma indica che tutte le operazioni di uscita (PRINT) seguenti devono essere scritte alla fine dell'archivio specificato. L'effetto della primitiva è semplicemente di posizionarsi alla fine dell'archivio. La primitiva APPEND deve essere seguita da una primitiva di scrittura.

Esempio:

Si abbia l'archivio indirizzi già creato, volendo aggiungere alcuni indirizzi, si definisce la primitiva:

AGGIUNGI$ = "APPEND INDIRIZZI"

e l'operazione di apertura è sostituita dall'istruzione:

PRINT D$; AGGIUNGI$

Prima di presentare il programma, è necessario modificare il programma di scrittura iniziale, in modo tale da non dover scrivere il nome "ZZ".

Si ottiene allora:

```
 1   REM SCRITTURA DI INDIRIZZI
 5   DIM N$ (100), IN$ (100)
10   D$ = CHR$ (4)
20   PRINT D$; "OPEN INDIRIZZO"
30   SCRIVI$ = "WRITE INDIRIZZO"
35   REM
36   REM INSERIMENTO DEGLI INDIRIZZI
37   REM
40   FOR I = 1 TO 100
50   INPUT "NOME COGNOME"; N$ (I)
```

```
 55     IF N$ (I) = "ZZ" THEN 80
 60     INPUT "INDIRIZZO"; IN$ (I)
 65     PRINT N$
 70     NEXT I
 74     REM
 75     REM SCRITTURA SU DISCO
 76     REM
 80     PRINT D$; SCRIVI$
 90     FOR J = 1 TO I - 1
100     PRINT N$ (J)
105     PRINT IN$ (J)
110     NEXT J
120     PRINT D$; "CLOSE INDIRIZZO"
130     END
```

Il programma per aggiungere nuovi indirizzi è:

```
  1     REM AGGIUNTA DI INDIRIZZI
        IN UN ARCHIVIO SEQUENZIALE
  5     DIM N$ (100), IN$ (100)
 10     D$ = CHR$ (4)
 20     AGGIUNGI = "APPEND INDIRIZZO"
 25     PRINT D$; AGGIUNGI$
 30     SCRIVI$ = "WRITE INDIRIZZO"
 35     REM
 36     REM INSERIMENTO DEGLI INDIRIZZI
 37     REM
 40     FOR I = 1 TO 100
 50     INPUT "NOME COGNOME"; N$ (I)
 55     IF N$ (I) = "ZZ" THEN 80
 60     INPUT "INDIRIZZO"; IN$ (I)
 65     PRINT N$
 70     NEXT I
 74     REM
 75     REM SCRITTURA SU DISCO
 76     REM
 80     PRINT D$; SCRIVI$
 90     FOR J = 1 TO I - 1
100     PRINT N$ (J)
105     PRINT IN$ (J)
110     NEXT J
120     PRINT D$; "CLOSE INDIRIZZO"
130     END
```

Esempio di esecuzione:

```
]RUN

NOME COGNOME DANTE ALIGHIERI
INDIRIZZO FIRENZE

NOME COGNOME LUIGI GALVANI
INDIRIZZO BOLOGNA

NOME COGNOME ZZ
```

I due nomi precedenti sono stati aggiunti alla fine dell'archivio.

Il programma di lettura modificato verifica il termine dell'archivio per mezzo dell'istruzione ONERR GO TO 80. Se il codice ottenuto da una istruzione PEEK (222) è uguale a 5, si è arrivati alla fine dell'archivio. Si ottiene allora:

```
  1   REM LETTURA DEGLI INDIRIZZI
  5   DIM N$ (100), IN$ (100)
 10   3^6 CHR$ (4)
 20   PRINT D$; "OPEN INDIRIZZO"
 30   LEGGI$ = "READ INDIRIZZO"
 34   PRINT D$; LEGGI$
 35   REM
 36   REM LETTURA SU DISCO
 37   REM
 40   FOR I = 1 TO 100
 45   ONERR GO TO 80
 50   INPUT N$ (I)
 60   INPUT IN$ (I)
 70   NEXT I
 74   REM
 75   REM STAMPA DEGLI INDIRIZZI
 76   REM
 80   C = PEEK (222)
 85   IF C = 5 THEN 90
 86   PRINT "ERRORE DOS NO"; C
 90   FOR J = 1 TO I - 1
100   PRINT N$ (J)
105   PRINT IN$ (J)
106   PRINT
110   NEXT J
120   PRINT D$; "CLOSE INDIRIZZO"
130   END
```

Esecuzione

]RUN

GIUSEPPE GARIBALDI
CAPRERA

UGO FOSCOLO
ZANTE

ALESSANDRO MANZONI
MILANO

CARLO LEVI
EBOLI

ENRICO FERMI
ROMA

DANTE ALIGHIERI
FIRENZE

LUIGI GALVANI
BOLOGNA

Si vede dunque che gli ultimi due nomi sono stati aggiunti alla fine dell'elenco.

Posizionamento in un archivio sequenziale

Questa primitiva permette di posizionarsi non importa dove nell'archivio, in vista di una lettura o di una scrittura. La posizione è identificata da uno spostamento relativo alla posizione attuale di un registratore che identifica l'item corrente. È possibile quindi il suo spostamento di K items più avanti.
La sintassi della primitiva è:

POSITION NOME ARCHIVIO, R K

R è l'iniziale della parola Record (registrazione), mentre K è un intero che precisa il numero di items da saltare.

Così, per esempio:

Se K = 0 (o senza nessuna indicazione) verrà letto o scritto l'item corrente.
Se K = 1 verrà letto o scritto l'item seguente
Se K = *n* verrà letto o scritto l'*n*-simo item più avanti.

Se il valore di K è tale che non esistono altri items nello schedario, allora si ottiene un messaggio d'errore.

L'istruzione di posizionamento deve precedere l'istruzione di lettura o di scrittura dell'archivio.

Esempio:

Nell'archivio INDIRIZZI precedente, vogliamo scrivere un programma che legga solo i nomi delle persone, ignorandone l'indirizzo.

Definiamo una primitiva AVANCE$ e nella istruzione 60 definiamo un posizionamento di 1, cioè un salto di registro.

Si ottiene allora il seguente programma:

```
  1   REM LETTURA DEGLI INDIRIZZI
  5   DIM N$ (100), IN$ (100)
 10   D$ = CHR$ (4)
 20   PRINT D$; "OPEN INDIRIZZO"
 25   AVANZA$ = "POSITION INDIRIZZO", R
 30   LEGGI$ = "READ INDIRIZZO"
 35   REM
 36   REM LETTURA SU DISCO
 37   REM
 40   FOR I = 1 TO 100
 45   ONERR GO TO 80
 46   PRINT D$; LEGGI$
 50   INPUT N$ (I)
 60   PRINT D$; AVANZA$; 1
 70   NEXT I
 74   REM
 75   REM STAMPA DEGLI INDIRIZZI
 76   REM
 80   C = PEEK (222)
 85   IF C = 5 THEN 90
 86   PRINT "ERRORE DOS NO"; C
 90   FOR J = 1 TO I - 1
100   PRINT N$ (J)
106   PRINT
110   NEXT J
120   PRINT D$; "CLOSE INDIRIZZO"
130   END
```

```
]RUN
GIUSEPPE GARIBALDI
UGO FOSCOLO
ALESSANDRO MANZONI
CARLO LEVI
ENRICO FERMI
DANTE ALIGHIERI
LUIGI GALVANI
```

Utilizzazione di un archivio in modo carattere

Fino ad ora avevamo visto che un archivio sequenziale era costituito da items separati da carattere ®.

La primitiva POSITION permette di posizionarci all'inizio degli items e il parametro precisato in questa primitiva permette di saltare un certo numero di items.

Ciò non è abbastanza flessibile, specialmente volendo trattare gli items in modo carattere.

In questo caso bisogna poter disporre di un meccanismo che permetta di posizionarci in qualsiasi punto dell'archivio. Ciò è possibile grazie all'utilizzo di un sistema di puntatori che indicano il punto esatto in cui si vuole leggere o scrivere nell'archivio.

In questo caso, le primitive WRITE e READ devono essere associate a un puntatore che indica la posizione esatta, in rapporto all'inizio dell'archivio, del punto in cui si vuole scrivere.

La sintassi di queste primitive è:

WRITE NOME ARCHIVIO, B_n

B è il carattere che specifica il puntatore di carattere (B sta per «Byte»). e *n* è la posizione esatta in numero di caratteri, in funzione dell'inizio dell'archivio, in cui ci si vuole posizionare in scrittura.

L'inizio dell'archivio corrisponde alla posizione 0.

I	carattere	0
T	—	1
E	—	2
M	—	3
1	—	4
®	—	5
I	—	6
T	—	7
E	—	8
M	—	9
2	—	10
®	—	11
I	—	12
T	—	13
E	—	14
M	—	15
3	—	16
®	—	17
FINE ARCHIVIO		

Esempio:

WRITE ARCH, B 14

indica che si vuole scrivere nell'archivio, di nome ARCH, a partire dal 15° carattere.

Nota. — Il valore del puntatore è un valore assoluto che deve tener conto dei caratteri ®, dei bianchi e di tutti gli altri caratteri presenti fisicamente nell'archivio.

Con una tale primitiva, bisogna dunque fare attenzione, in quanto si possono schiacciare oltre un determinato item le informazioni, o addirittura oltre

l'archivio considerato, se il puntatore fa riferimento a un carattere che si trova oltre la fine dell'archivio.

Consigliamo il principiante di essere prudente con questa istruzione.

D'altra parte è ben preferibile prendere familiarità con questo tipo di istruzione, utilizzando la primitiva di lettura la cui sintassi è analoga:

READ NOME ARCH, B_n
(LEGGI)

I parametri hanno lo stesso significato che per la primitiva WRITE.

Esempio:

READ ARCH, B52

permette di posizionarsi al 53° carattere dell'archivio ARCH per la lettura.

Comunque, la primitiva READ (LEGGI) non ha come effetto la lettura e deve essere seguita da un'istruzione di entrata (INPUT) che leggerà i dati, a partire dalla posizione definita dal puntatore.

Applicazione: creazione di un sistema di archivi indicizzati

La possibilità di lavorare in modo carattere permette di creare un sistema di archivi indicizzati che utilizzano le primitive SGAS.

Organizzazione di un archivio di questo tipo

Un archivio indicizzato è un archivio scomposto in articoli scritti sequenzialmente su dischi, ai quali si può aver accesso con l'intermediazione di un indice.

Un *indice* è una tabella che viene consultata sequenzialmente e che indica la posizione di ciascun articolo nell'archivio.

Può essere schematizzata da una serie di sinonimi:

NOME ARTICOLO   INDIRIZZO ARTICOLO

Il nome dell'articolo può eventualmente essere un numero e l'indirizzo dell'articolo è la posizione, in numero di caratteri (ottetti), relativa all'inizio dell'archivio.

Se l'indice è di lunghezza prefissata, può essere inserito all'inizio dell'archivio.

Se l'indice è di lunghezza variabile, lo si deve mettere alla fine. Questa soluzione è preferibile, in quanto è più generale, ma ciò implica una grande si-

curezza di funzionamento, poichè, se non si può scrivere di nuovo l'indice, o se lo si è mal riscritto, l'archivio risulta difficilmente recuperabile.

La struttura di un archivio indicizzato può essere rappresentata nel modo seguente:

INDIRIZZO INDICE	
1° articolo	
2° articolo	
.....	
n° articolo	
N° ART IND	
N° ART IND	
	INDICE
N° ART IND	

Schema di un archivio indicizzato

Nota. — Se l'indice contiene nomi di articoli o numeri, gli articoli non sono obbligatoriamente scritti nell'ordine, 1°, 2°... ma a seconda del bisogno.

Le primitive associate

Le primitive non esistono sui SGAD dei microelaboratori presi in considerazione, tuttavia li si può sviluppare grazie alla possibilità di definire dei puntatori.

Dal punto di vista del SGAD, gli archivi corrispondenti sono archivi sequenziali.

L'indice dovrà dunque essere mantenuto per programma.

Le primitive da sviluppare saranno:

- la CREAZIONE di un indice;
- la SCRITTURA di un articolo;
- la LETTURA di un articolo;

Poichè la lettura di un articolo implica la lettura dell'indice in memoria centrale, si può prevedere una primitiva di APERTURA di un archivio indicizzato, che consisterà nella lettura dell'indice in memoria.

Allo stesso modo, poichè la scrittura di un articolo implica una messa in evidenza e una scrittura dell'indice, potrà essere introdotta da una primitiva di CHIUSURA dell'archivio indicizzato: ciò avrà per effetto la scrittura dell'indice.

Se l'archivio può essere variato, si può ugualmente prevedere una primitiva di CANCELLAZIONE di un articolo.

Per quanto riguarda la cancellazione di un archivio, non è necessario prevedere questo tipo di operazione, in quanto la cancellazione di un archivio sequenziale è già prevista per conto suo.

Queste primitive sono lasciate alla programmazione del lettore, a titolo d'esercizio.

GLI ARCHIVI AD ACCESSO DIRETTO (ALEATORIO)

È possibile un altro metodo di accesso: si tratta del metodo di accesso diretto (detto anche aleatorio, benchè questo nome sia improprio).

In questo caso, l'archivio è organizzato in articoli o in registri, come per l'accesso indicizzato.

Tuttavia, qui, non esistono, propriamente parlando, degli indici, ed è il numero di registrazione che permette di accedere direttamente ad un articolo.

Ciò implica che gli articoli siano di lunghezza fissa e che venga riservato dello spazio per gli articoli che non sono ancora stati scritti.

L'accesso è chiamato diretto poichè la posizione di un articolo può essere ottenuta direttamente con un semplice calcolo:

INDIRIZZO DI PARTENZA =
N° ARTICOLO × LUNGHEZZA IN CARATTERI

Il vantaggio di tale metodo è la rapidità d'accesso; l'inconveniente è la perdita di spazio causato dagli articoli non ancora scritti ed, eventualmente, dalla rigidità imposta dalla lunghezza fissa degli articoli.

Le primitive di un SGAD in accesso diretto

Si hanno ancora delle primitive del SGAD sequenziale, ma con parametri supplementari.

APERTURA in accesso diretto

Il parametro che deve essere specificato ulteriormente è la lunghezza dell'articolo.

La sintassi della primitiva è allora:

OPEN NOME ARCHIVIO, L_n
(APRI)

L definisce il parametro lunghezza e *n* è un numero intero da 1 a 32767, che rappresenta il numero di caratteri fissato per un articolo.

Nota. – In apertura si possono ugualmente precisare i parametri abituali che definiscono il disco, il volume e il controllo (cfr. schedari sequenziali).

Il valore, preso per difetto, di L è 1. Bisogna dunque conoscere la lunghezza degli articoli per utilizzare un archivio ad accesso diretto.

Esempio:

```
10      D$ = CHR$ (4)
20      PRINT D$ "OPEN INDIRIZZI, L 100"
.....   ...........
```

permette di aprire un archivio indirizzi avente dei registri di 100 caratteri, cioè con 5 righe di 20 caratteri per ogni indirizzo.

LETTURA/SCRITTURA in accesso diretto

Bisogna precisare un parametro necessario all'accesso diretto, per sapere il numero dell'articolo. Si hanno dunque le seguenti primitive:

READ NOME ARCHIVIO, R_n, B_n
(LEGGI)
WRITE NOME ARCHIVIO, R_n, B_n
(SCRIVI)

R_n è il parametro che precisa l'articolo (Record) da leggere o scrivere. B_n è un parametro facoltativo che abbiamo già visto per gli archivi sequenziali, ma che qui rappresenta l'*n*-simo carattere a partire dal quale bisognerà leggere o scrivere nell'articolo specificato. Si tratta dunque di un puntatore all'interno dell'articolo stesso. Bisogna avere molta precauzione nel suo uso, specialmente in scrittura.

Applicazione. – Si debba formare un archivio di indirizzi.

Si supponga che gli individui siano identificati da un unico numero. Si limiti la dimensione di un articolo indirizzo a 100 caratteri.

Per precisare le primitive di lettura-scrittura di un articolo *n*, è sufficiente definire una catena di caratteri come:

"WRITE NOME ARCH, R"

associandogli un intero N che seguirà immediatamente questa catena di caratteri nell'istruzione corrispondente, ad esempio:

PRINT D$; "WRITE NOME ARCH, R"; N

Ciò indicherà che si vuole scrivere il registro N, chiamato NOME ARCH, dell'archivio.

Si ottiene allora il programma seguente, che comincia con lo scrivere gli indirizzi e, successivamente, permette di rileggerli, precisando il numero del registro o dell'articolo.

Programma di scrittura

```
  1     REM ARCHIVIO INDIRIZZI AD ACCESSO DIRETTO
  4     DIM  N (100)
  5     DIM  N$ (100), IN$ (100)
 10     D$ = CHR$ (4)
 15     APRI$ = "OPEN ACDIR,L100"
 16     CHIUDI$ = "CLOSE ACDIR"
 20     PRINT D$; APRI$
 30     SCRIVI$ = "WRITE ACDIR, R"
 35     REM
 36     REM INSERIMENTO DEI DATI
 37     REM
 40     FOR I = 1 TO 100
 45     INPUT "NUMERO"; N (I)
 50     INPUT "NOME COGNOME"; N$ (I)
 55     IF N$ (I) = "ZZ" THEN 80
 60     INPUT "INDIRIZZO"; IN$ (I)
 65     PRINT N$
 70     NEXT I
 74     REM
 75     REM SCRITTURA SU DISCO
 80     REM
 90     FOR J = 1 TO I-1
 95     PRINT D$; SCRIVI$; N (J)
100     PRINT N$ (J)
105     PRINT IN$ (J)
110     NEXT J
120     PRINT D$; CHIUDI$
130     END
```

```
RUN

NUMERO 1

NOME COGNOME PIERRE DUPON

INDIRIZZO PARIGI
```

NUMERO 1

NOME COGNOME PIERRE DUPONT
INDIRIZZO PARIGI

NUMERO 4

NOME COGNOME MARIO BIANCHI
INDIRIZZO ROMA

NUMERO 9

NOME COGNOME JOHN BROWN
INDIRIZZO LONDRA

NUMERO 17

NOME COGNOME ZZ

Il programma seguente permette di leggere un articolo di un archivio, di correggerlo e di scriverlo di nuovo.

```
 1    REM ARCHIVIO INDIRIZZI IN ACCESSO DIRETTO
 2    REM
 3    REM PRIMITIVE DEL SGAD
 4    REM
10    D$ = CHR$ (4)
15    APRI$ = "OPEN ACDIR,L100"
16    CHIUDI$ = "CLOSE ACDIR"
20    SCRIVI$ "WRITE ACDIR,R"
30    LEGGI$ = "READ ACDIR,R"
35    REM
36    REM INSERIMENTO DEL NUMERO DI ARTICOLO K
37    REM
40    INPUT "NUMERO"; K%
45    IF K% < = 0 THEN 230
50    ONERR GO TO 190
52    PRINT D$; APRI$
55    PRINT D$; LEGGI$; K%
56    REM
57    REM LETTURA DELL'ARTICOLO SU DISCO
58    REM
60    INPUT N$: INPUT IN$
65    REM
66    REM STAMPA DELL'INDIRIZZO
67    REM
70    PRINT: PRINT N$: PRINT IN$
72    PRINT
```

```
75      PRINT D$; CHIUDI$
76      REM
77      REM CORREZIONE DELL'ARTICOLO
78      REM
80      INPUT "NUOVA SCRITTURA O/N?"; R$
90      IF R$ = "0" THEN 130
95      GO TO 40
100     REM
110     REM SCRITTURA DI UN ARTICOLO
120     REM
130     INPUT "NOME COGNOME"; N$
140     INPUT "INDIRIZZO"; IN$
150     PRINT D$; APRI$
160     PRINT D$; SCRIVI$; K%
170     PRINT N$; PRINT IN$
180     PRINT D$; CHIUDI$
185     GO TO 220
186     REM
187     REM CASO DI ERRORE VERIFICARE LA FINE DI AR-
        CHIVIO O LA MANCANZA D'ARTICOLO
188     REM
190     C = PEEK (222)
200     IF C = 5 THEN INPUT "SCRITTURA O/N?"; R$: GO TO
        90
210     PRINT "ERRORE DOS NO"; C
220     GO TO 40
230     END
```

Esercizi

1. *Scrivere un programma che permetta di creare un indice in un archivio sequenziale.*
   Per fare ciò, il primo articolo del programma sequenziale sarà considerato come un numero di 5 cifre (da 0 a 99999) che indicherà l'indirizzo dell'indice.
   Si abbia:

×	
×	
×	
×	INDIRIZZO
×	INDICE
®	

L'indice stesso è composto da più registri, il primo dei quali rappresenta il numero di articoli dell'archivio su tre cifre (da 0 a 999)

INDICE		
	N	
	N	NUMERO DI ARTICOLI
	N	

2. *Scrivere un programma che scriva un nuovo articolo in un archivio indicizzato (organizzato sequenzialmente); ogni ingresso di un articolo nell'indice è composto da un nome di articolo su tre caratteri (un numero di tre cifre), seguito dall'indirizzo dell'inizio dell'articolo su 5 cifre.*

N	
O	NOME DELL'ARTICOLO
M	
A	
D	INDIRIZZO DELL'ARTICOLO
R	
E	
S	
®	

3. *Scrivere un programma che legga un articolo da un archivio indicizzato.*

CONCLUSIONE

In questo capitolo abbiamo presentato i concetti fondamentali necessari per il trattamento di dati strutturati negli archivi. Benchè i programmi siano semplici, essi rappresentano differenti modi di trattamento disponibili su tutti i sistemi. Le primitive presentate sono analoghe su tutti i sistemi, compresi i più grossi. Certo, i sistemi di archivi devono essere più elaborati se si vuole prendere in considerazione la nozione di spartizione di archivi tra utenti, oltre allo sviluppo dei sistemi di base dei dati.

Nondimeno, gli strumenti disponibili attualmente sui più piccoli sistemi a dischi flessibili sono sufficientemente elaborati, in modo tale da poter permet-

tere agli utenti di sviluppare dei sistemi di immagazzinaggio di dati sotto forma di piccole basi di dati, poichè, in questo caso, non si pongono problemi di spartizione simultanea degli archivi o della base di dati. Lo sviluppo delle tecniche dei sistemi di risposta permetterà, in un prossimo futuro, di rendere accessibili i dati immagazzinati nei più piccoli sistemi, partendo da qualunque terminale o da qualunque sistema connesso ad una rete.

CAPITOLO 6

IL TRATTAMENTO GRAFICO IN BASIC

In questo capitolo, studieremo le possibilità di fare grafici, utilizzando il BASIC. Benchè ciò non faccia parte del linguaggio standard, esistono numerosi sistemi BASIC che utilizzano istruzioni, con le quali si possono trattare problemi grafici. Fino a qualche anno fa, era necessario acquistare delle consoles di visualizzazione grafica relativamente costose. Ora possibilità simili esistono su sistemi connessi ad apparecchi televisivi standard. In particolare, i programmi presentati in questo capitolo, sono stati provati su un sistema APPLE che permette anche di lavorare con il colore.

Vedremo che le possibilità grafiche aumentano il campo di utilizzazione del linguaggio. Possiamo dire che è un peccato che le istruzioni grafiche non siano meglio standardizzate, in quanto, dal punto di vista pedagogico, è sovente facile e più piacevole far comprendere i concetti della programmazione con programmi che visualizzino ciò che stanno elaborando.

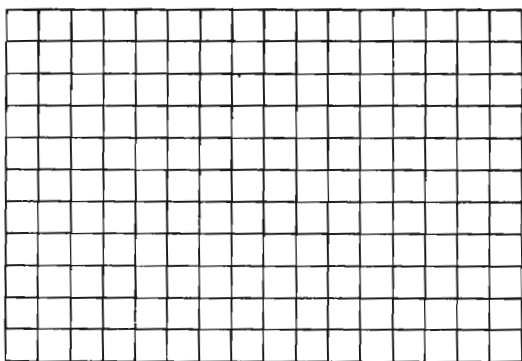
Indipendentemente dalle particolarità del sistema utilizzato, il capitolo sarà incentrato sui problemi generali incontrati in tema di grafici: cambio di origine, cambio di scala, trattamento dei vettori.

In questa maniera, un utente che non disponga dello stesso sistema, può, senza difficoltà, applicarlo ad un altro.

1. INTRODUZIONE AL TRATTAMENTO GRAFICO IN BASIC

Benchè il linguaggio BASIC standard non disponga di istruzioni per i grafici, esistono diversi sistemi di microelaboratori che usano tali facilitazioni, specialmente quando il microelaboratore è realizzato in unione con un televisore a colori. Il sistema APPLE, che prenderemo come esempio nel seguito del capitolo, è di questo tipo. È infatti il sistema più conosciuto per elaborare grafici, utilizzando un banale televisore.

Il passaggio al modo grafico

Il modo grafico presuppone che lo schermo non sia più diviso in linee di caratteri, ma in una griglia di punti, a cui ci si indirizza attraverso un sistema di coordinate cartesiane: l'ascissa (X) e l'ordinata (Y).

Griglia di punti su uno schermo

L'uso dello schermo come griglia di punti (in effetti sono dei quadrati), è specificato da un comando speciale.

Sul sistema APPLE esistono due modi, corrispondenti a differenti griglie e a differenti possibilità di colore.

Il primo modo è il modo grafico semplice, che utilizza una griglia di 40 x 40 e la possibilità di sedici differenti colori. Questo modo è contraddistinto dal comando:

GR (Grafico)

Nel modo grafico semplice, la parte in basso dello schermo è disponibile per la visualizzazione di quattro linee di testo. Il passaggio per l'uso della totalità dello schermo per visualizzare un testo viene fatto attraverso il comando: TEXT. Esiste poi un comando per utilizzare tutto lo schermo in modo grafico.

Per altro, questi comandi possono essere inseriti in un programma BASIC.

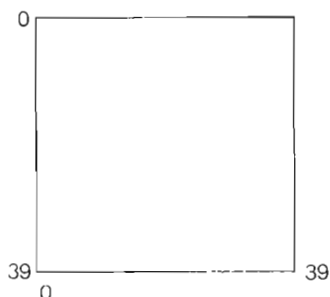
Esiste poi il modo grafico di *alta risoluzione*, che utilizza una griglia di 280 x 160, con una parte dello schermo riservata al testo, con la quale si può arrivare ad una griglia di 280 x 192. Nel primo caso, cioè con il testo, si utilizza il comando:

HGR (grafico ad alta risoluzione)

Nel secondo caso, si utilizza il comando:

HGR2 (grafico ad alta risoluzione su tutto lo schermo)

2. IL MODO GRAFICO SEMPLICE

La griglia utilizzata è di 40 x 40 e ciascun punto è determinato da coordinate che vanno da 0 a 39. Il punto di coordinate (0,0) è il punto più in alto e a sinistra dello schermo.

Si noterà che le ascisse vanno crescendo da sinistra verso destra e le ordinate vanno crescendo dall'alto in basso, contrariamente all'uso abituale che se ne fa in matematica.

IL COLORE

Prima di vedere come si può visualizzare un punto su uno schermo, definiamo la gamma dei colori disponibili. Nel sistema APPLE, i colori disponibili sono definiti da un codice numerico che va da 0 a 15:

0	Nero	8	Bruno
1	Bruno scuro	9	Arancione
2	Blu scuro	10	Grigio
3	Violetto lavanda	11	Rosa
4	Verde scuro	12	Verde
5	Grigio scuro	13	Giallo
6	Blu	14	Blu acqua
7	Blu cielo	15	Bianco

La definizione di un colore si fa con l'aiuto dell'istruzione COLORE (COLOR):

$$\text{COLOR} = n \qquad 0 < n \leq 15$$

Allo stesso modo, è possibile definire il colore con una espressione aritmetica: COLOR = espressione aritmetica. In questo caso il colore associato sarà ottenuto attraverso il valore corrispondente alla parte intera dell'espressione modulo 16.

Visualizzazione di un punto

Si tratta di un comando di uscita che precisa le coordinate del punto che si desidera visualizzare. Questa istruzione si trova in diversi sistemi BASIC in versioni abbastanza simili. Si tratta dell'istruzione DISEGNA (PLOT).

La sua sintassi è:

```
PLOT     X, Y
```

dove X e Y sono le variabili o le costanti che rappresentano le coordinate del punto da visualizzare. Il colore del punto sarà determinato dall'ultimo valore definito dal comando COLOR. Se questo non fosse stato impartito, il valore, preso per difetto, sarà lo 0 (nero).

Esempio di programma di visualizzazione di colori

```
10    GR
20    FOR I = 0 TO 15
30    COLOR = I
40    PLOT I,I
50    NEXT I
60    END
```

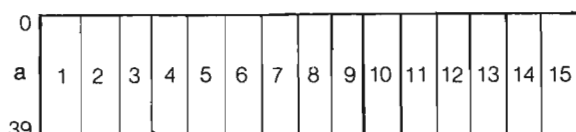
Questo programma visualizza i 16 colori sui primi 16 punti della diagonale principale della griglia.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Nero															
1		Magenta														
2			Blu scuro													
3				Lavanda												
4					Verde scuro											
5						Grigio scuro										
6							Blu									
7								Blu cielo								
8									Bruno							
9										Arancione						
10											Grigio chiaro					
11												Rosa				
12													Verde			
13														Giallo		
14															Acqua	
15																Bianco

Visualizzazione di bande verticali di colore

Il programma precedente può essere modificato per dare delle bande verticali:

```
10    GR
20    FOR I = 0 TO 15
30    COLOR = I
40    FOR J = 0 TO 39
50    PLOT I, J
60    NEXT J
70    NEXT I
```

In questo caso si ottiene:

15 bande di colore

Riconoscimento del colore di un punto attraverso programma

Può essere necessario di voler conoscere il colore di un punto di coordinate X, Y. Ciò è possibile grazie alla funzione SCRN (X, Y), in cui X e Y possono avere qualunque valore intero da 0 a 39 (o 47 se si utilizza tutto il video).

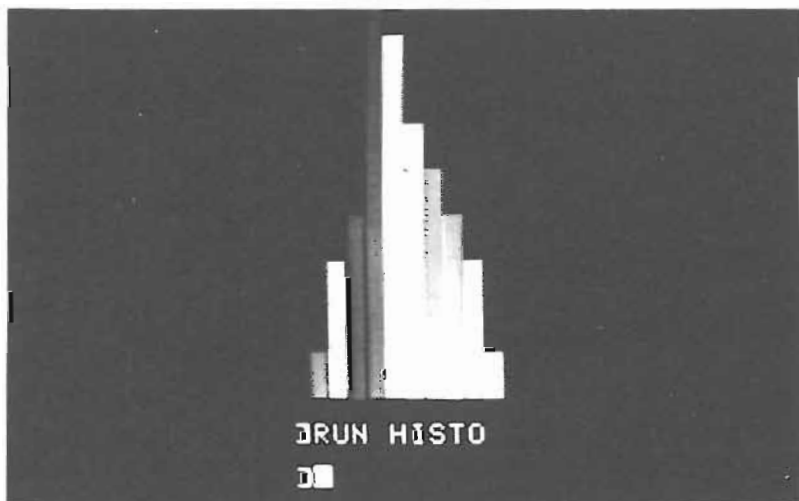
Il valore di ritorno, attraverso la funzione SCRN, è un intero compreso tra 0 e 15.

Applicazione alla rappresentazione di un istogramma

Si debba rappresentare sotto forma di istogramma una popolazione di individui divisi in 10 classi, in modo che la classe più numerosa utilizzi l'altezza massima dello schermo.

Si ottiene il seguente programma:

```
10    DIM C (10), H (10)
15    REM LETTURA DEI DATI IN CIASCUNA CLASSE
20    FOR I = 1 TO 10
30    READ C (I)
40    NEXT
50    DATA 10,30,40,100,80,60,50,40,30,10
```

```
 55   REM RICERCA DELLA CLASSE MASSIMA
 60   FOR I = 1 TO 10
 70   IF C (I) > C (I - 1) THEN IM = I : M = C (I)
 80   NEXT I
 90   HM = 39
 95   REM CALCOLO DELLA SCALA
100   SC = HM/M
105   REM CALCOLO DELL'ALTEZZA
      DI CIASCUNA CLASSE
110   FOR I = 1 TO 10
120   H (I) = INT (SC * C (I))
130   NEXT I
140   GR
145   REM VISUALIZZAZIONE DELL'ISTOGRAMMA
150   FOR I = 1 TO 10
160   COLOR I + 1
170   FOR J = 39 TO 39 - H (I) STEP -1
180   PLOT I, J
190   NEXT I, J
200   END
```

Rimozione di un punto sullo schermo

Sino ad ora abbiamo visto gli ordini di visualizzazione statica. Può essere però interessante far muovere un punto sul video.

Per fare ciò, bisogna cambiare le coordinate del punto e, se si desidera visualizzare la traiettoria del punto stesso, non c'è nient'altro da fare. Se, al

contrario, non si vuole visualizzare la traiettoria, ma semplicemente la posizione del punto in funzione del tempo, si deve cancellare la precedente posizione. Ciò permette, in particolare, di fare dell'animazione.

Esempi:

```
 10    GR
 20    COLOR = 10
 30    INPUT "ORIGINE"; X, Y
 40    PLOT, X, Y
 50    DX = 1
 60    DY = 1
 70    FOR I = 1 TO 10000
 80    X = X + DX
 90    Y = Y + DY
100    IF  X  >  39  THEN  X  =  0
110    IF Y > 39  THEN  Y  =  0
120    PLOT X, Y
130    NEXT I
200    END
```

In questo programma, il punto si muove lasciando una traccia della sua traiettoria. Quando il punto arriva al bordo dello schermo, appare dall'altro lato. Tuttavia, una volta che la traiettoria è tracciata, non si vede più il movimento.

Volendo ottenere solo il movimento di un punto, si sostituiranno le istruzioni dal 70 al 130 con il seguente programma:

```
 70    X1 = X + DX
 80    Y1 = Y + DY
 90    IF X1 > 39 THEN X1 = 0
100    IF Y1 > 39 THEN Y1 = 0
110    COLOR = 10
120    PLOT X1, Y1
130    COLOR = 0
140    PLOT X, Y
150    X = X1
160    Y = Y1
170    GO TO 70
180    END
```

Nota. — In modo più generale, si può rimuovere un punto di colore *n*, in modo tale che si possa posizionare su un fondo di colore differente.

Così, l'inizio del programma potrebbe essere:

```
INPUT "COLORE DEL FONDO"; F
INPUT "COLORE DEL FUNTO"; P
COLOR = F
FOR I = 0 TO 39
FOR J = 0 TO 39
PLOT I, J
NEXT I, J
```

L'istruzione 110 diventerà : COLOR = P
e l'istruzione 130 : COLOR = F

```
1     INPUT "COLORE DEL FONDO"; F
2     INPUT "COLORE DEL PUNTO"; P
3     GR: COLOR = F
4     FOR I = 0 TO 39
5     FOR J = 0 TO 39
6     PLOT I, J
7     NEXT I, J
10    COLOR = P
20    HOME
30    INPUT "ORIGINE"; X, Y
40    PLOT X, Y
50    DX = 1
60    DY = 1
70    X1 = X + DX
80    Y1 = Y + DY
90    IF X1 > 39 THEN X1 = 0
100   IF Y1 > 39 THEN Y1 = 0
110   COLOR = P: PLOT X1, Y1
120   COLOR = F: PLOT X, Y
130   X = X1 : Y = Y1
140   GO TO 70
200   END
```

Il gioco della vita

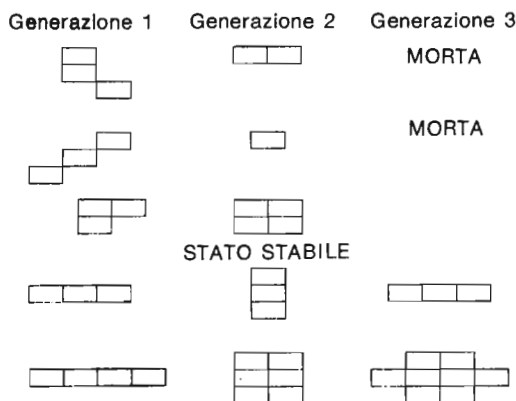
Il matematico CONWAY ha proposto un gioco detto «Gioco della Vita», considerando delle cellule capaci di riprodursi, di morire o di sopravvivere, obbedendo a certe regole dette «genetiche». Queste cellule sono rappresentate da elementi su una scacchiera, la cui grandezza può essere arbitraria.

Ciascuna cellula è dunque circondata da otto caselle che possono contenere altre cellule.

Le regole sono le seguenti:

1. La sopravvivenza: ciascuna cellula che ha due o tre cellule adiacenti sopravvive fino alla prossima generazione.
2. La morte: ciascuna cellula avente quattro o più cellule adiacenti muore per sovrapopolazione. Ogni cellula che ha una o zero cellule adiacenti muore di isolamento.
3. La nascita: ciascuna casella esattamente adiacente a tre cellule fa nascere una nuova cellula per la generazione seguente.

È importante notare che tutte le nascite e le morti hanno luogo nello stesso tempo nel corso di una generazione.

Esempio:

```
10        DIM C (40, 40)
20        REM NR DELLE CELLULE DI PARTENZA
30        INPUT "NR DELLE CELLULE"; N
35        GR: COLOR = 2
40        FOR I = 1 TO N
50        INPUT "COORDINATE ="; X, Y
60        PLOT X, Y
70        NEXT I
71        INPUT "LIMITI DELLO SCHERMO"; K, L: COLOR = 15
72        HLIN K - 2, L + 2 AT K - 2
73        HLIN K - 2, L + 2 AT L + 2
74        VLIN K - 1, L + 1 AT K - 2
75        VLIN K - 1, L + 1 AT L + 2
76        G = 1 : HOME
77        PRINT "GENERAZIONE"; G
80        FOR I = K TO L
```

```
90    FOR J = K TO L
100   NC = 0
110   FOR I1 = I – 1 TO I + 1
120   FOR J1 = J – 1 TO J + 1
130   IF SCRN (I1, J1) < >0 THEN NC = NC + 1
140   NEXT J1
150   NEXT I1
160   IF SCNR (I, J) = 0 THEN 200
170   NC = NC –1
175   REM MORTE PER SOVRAPOPOLAZIONE
180   IF NC > 4 THEN C (I, J) = 0: GO TO 290
185   REM SOPRAVVIVENZA ALLA PROSSIMA
      GENERAZIONE
190   IF NC > = 2 THEN C (I, J) = 1 : GO TO 290
194   REM MORTE PER ISOLAMENTO
195   C (I, J) = 0
196   REM NASCITA
200   IF NC = 3 THEN C (I, J) = 1
290   NEXT J, I
300   REM VISUALIZZAZIONE DELLA NUOVA
      GENERAZIONE
310   FOR I = K TO L
320   FOR J = K TO L
330   IF C (I, J) = 0 THEN COLOR = 0 : GO TO 350
340   COLOR = 2
350   PLOT I, J
360   NEXT J, I
370   G = G + 1
375   HOME
380   PRINT "GENERAZIONE"; G
390   GO TO 80
500   END
```

Esercizi

1. *Modificare il programma di posizionamento di un punto sullo schermo; per farlo «rimbalzare» sui lati dello schermo stesso.*
2. *Scrivere un programma di movimento aleatorio.*
   Un punto è circondato da otto posizioni possibili, numerate da 1 a 8. Il movimento aleatorio consiste nel tirare a caso il «passo» successivo attraverso queste otto posizioni.

1	2	3
4		5
6	7	8

3. *Stesso esercizio, scegliendo un colore aleatorio per ciascun passo.*

RISOLUZIONI

N° 1

```
  1   INPUT "COLORE DI FONDO"; F
  2   INPUT "COLORE DEL PUNTO"; P
  3   GR: COLORE = F
  4   FOR I = 0 TO 39
  5   FOR J = 0 TO 39
  6   PLOT I, J
  7   NEXT J, I
 10   COLOR = P
 20   HOME
 30   INPUT "ORIGINE"; X, Y
 40   PLOT X, Y
 50   DX = 1
 60   DY = 1
 70   X1 = X + DX
 80   Y1 = Y + DY
 85   REM SE SI ARRIVA AL BORDO RIMBALZARE
 90   IF X1 = 0 OR X1 = 39 DX = – DX
100   IF Y1 = 0 OR Y1 = 39 DY = – DY
105   REM VISUALIZZARE IL NUOVO PUNTO
110   COLOR = P: PLOT X1, Y1
115   REM CANCELLARE IL PUNTO PRECEDENTE
120   COLOR = F : PLOT X, Y
130   X = X1 : Y = Y1
140   GO TO 70
200   END
```

N° 2

```
  1   INPUT "COLORE DEL FONDO"; F
  2   INPUT "COLORE DEL PUNTO"; P
  3   GR : COLOR = F
  4   FOR I = 0 TO 39
  5   FOR J = 0 TO 39
  6   PLOT I, J
  7   NEXT J, I
 10   COLOR = P
 20   HOME
 30   INPUT "ORIGINE"; X, Y
 35   PLOT X, Y
```

```
40      REM METTERE A ZERO LE TABELLE DX E DY
41      DIM DX (8), DY (8)
42      FOR I = 1 TO 8
43      READ DX (I), DY (I)
44      NEXT I
45      DATA-1,-1,0,-1,1,-1,-1,0,1,0,-1,1,0,1,1,1
50      REM SCELTA DELL'INCREMENTO ALEATORIO
60      K = INT (8 * RND (1) + 1)
70      X1 = X + DX (K)
80      Y1 = Y + DY (K)
85      REM SE SI ARRIVA AI BORDI PASSARE ALL'ALTRO
        LATO
90      IF X1 < 0  THEN X1 = 39
95      IF X1 > 39 THEN X1 = 0
100     IF Y1 < 0  THEN Y1 = 39
101     IF Y1 > 39 THEN Y1 = 0
105     REM VISUALIZZARE IL NUOVO PUNTO
110     COLOR = P : PLOT X1, Y1
115     REM CANCELLARE IL PUNTO PRECEDENTE
120     COLOR = F: PLOT X, Y
130     X = X1 : Y = Y1
140     GO TO 60
200     END
```

N° 3

```
  1     INPUT "COLORE DI FONDO"; F
  2     INPUT "COLORE DEL PUNTO"; P
  3     GR : COLORE = F
  4     FOR I = 0 TO 39
  5     FOR J = 0 TO 39
  6     PLOT I, J
  7     NEXT J, I
 10     COLOR = P
 20     HOME
 30     INPUT "ORIGINE"; X, Y
 35     PLOT X, Y
 40     REM METTERE A ZERO LE TABELLE DX E DY
 41     DIIM DX (8), DY (8)
 42     FOR I = 1 TO 8
 43     READ DX (I), DY (I)
 44     NEXT I
 45     DATA -1,-1,0,-1,1,-1,-1,0,1,0,-1,1,0,1,1,1
 50     SCELTA DELL'INCREMENTO ALEATORIO
```

```
60      K = INT (8 * RND (1) + 1)
70      X1 = X + DX (K)
80      Y1 = Y + DY (K)
85      REM SE SI ARRIVA AL BORDO PASSARE ALL'ALTRO
        LATO
90      IF X1 < 0  THEN X1 = 39
95      IF X1 > 39 THEN X1 = 0
100     IF Y1 < 0  THEN Y1 = 39
101     IF Y1 > 39 THEN Y1 =0
105     REM VISUALIZZARE IL NUOVO PUNTO
106     P = INT (15 * RND (1) + 1)
110     COLOR = P : PLOT X1, Y1
115     REM CANCELLARE IL PUNTO PRECEDENTE
120     COLOR = F: PLOT X, Y
130     X = X1 : Y = Y1
140     GO TO 60
200     END
```

La traccia di un segmento in grafico semplice

Si è visto che, con un programma, si poteva tracciare un segmento verticale.

Allo stesso modo, se si volesse tracciare un segmento orizzontale sulla dodicesima linea, si scriverà:

```
10      GR
15      COLOR = 10
20      FOR J = 0 TO 39
30      PLOT 12, J
40      NEXT J
50      END
```

Nella maggior parte dei sistemi grafici esistono delle istruzioni che permettono di realizzare tracce con una sola istruzione.

Traccia di un segmento orizzontale

Può essere realizzato con l'istruzione HLIN (linea orizzontale). La sua sintassi è:

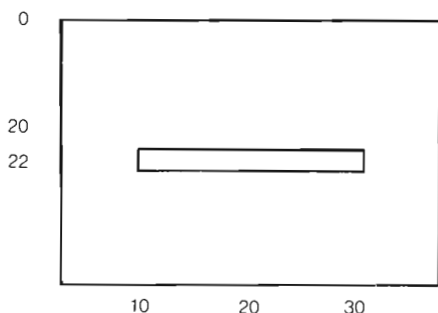
HLIN espressione 1, espressione 2 AT espressione 3

- espressione 1 rappresenta l'ascissa dell'origine del segmento (valore intero da 0 a 39);
- espressione 2 rappresenta l'ascissa della fine del segmento (valore intero da 0 a 39);

- espressione 3 rappresenta il valore dell'ordinata in cui deve essere tracciato il segmento orizzontale (deve essere compresa tra 0 e 39 o tra 0 e 47).

Esempio:

HLIN 10, 30 AT 22

Tracciare un segmento orizzontale di ordinata 22, dall'ascissa 10 all'ascissa 30

Traccia di un segmento verticale

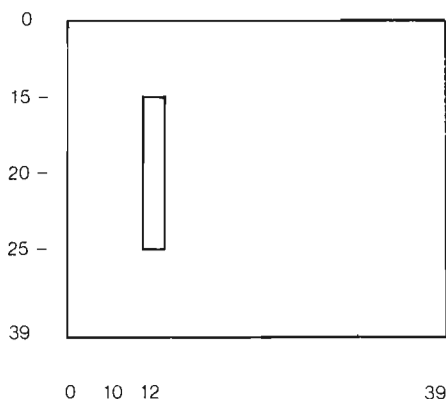
Una analoga istruzione permette di tracciare un segmento verticale. La sua sintassi è:

VLIN espressione 1, espressione 2 AT espressione 3

- espressione 1 ed espressione 2 rappresentano le ordinate degli estremi del segmento da tracciare;
- espressione 3 rappresenta l'ascissa del segmento verticale da visualizzare.

Esempio:

```
10    I = 15 : J = 25 : A = 12
20    GR : COLOR = 8
30    VLIN, I, J AT A
40    END
```

Questo programma permette di visualizzare un segmento verticale di ascissa 12 tra le ordinate 15 e 25

Traccia di un segmento punteggiato di direzione qualunque

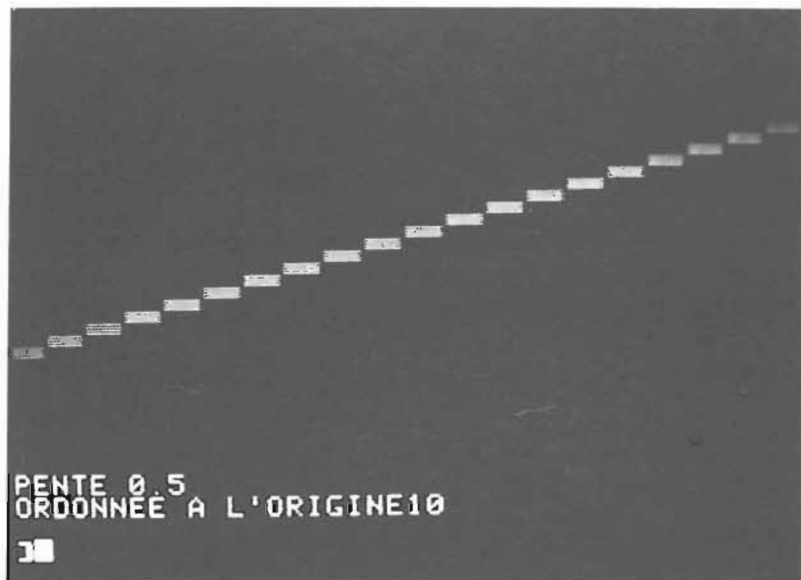
Si abbia la retta di equazione $y = ax + b$.

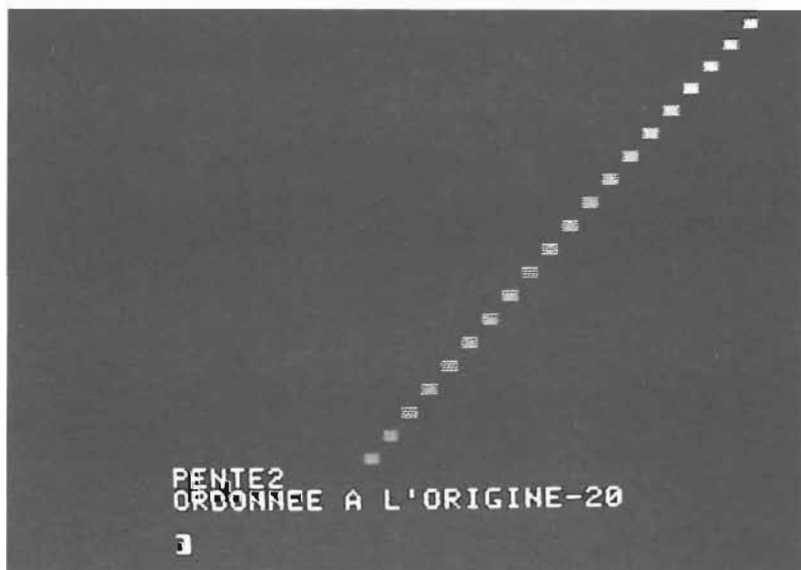
Nel modo grafico semplice, non esistono istruzioni che permettano di tracciare un segmento di direzione qualunque, in quanto la griglia dei punti è troppo grossolana. Si possono ottenere quindi solo delle tracce approssimate di una retta sotto forma di puntini. Il programma è:

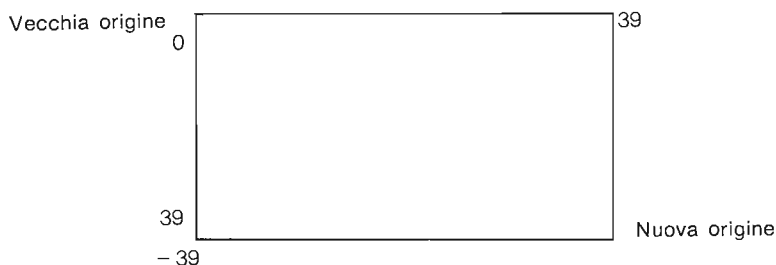
```
1     INPUT "PENDENZA"; A
11    INPUT "ORDINATA ALL'ORIGINE"; B
15    GR : COLOR = 5
20    FOR X = 0 TO 39
30    Y = 39 - INT (A * X + B)
35    IF Y < 0 OR Y > 39 THEN 50
40    PLOT X, Y
50    NEXT X
60    END
```

Nota. — Bisogna utilizzare Y = 39 — INT ($ax - b$), poichè l'asse delle y è preso nel senso contrario al senso abituale.

Esempio di esecuzione:

PENDENZA ? 0.5, ® Retta Y = X/2 + 10
ORDINATA ALL'ORIGINE ? 10 ®

Pendenza = 5
Ordinata all'origine 10

PENDENZA ? 2, ® Retta Y = 2X − 20
ORDINATA ALL'ORIGINE? − 20 ®

Pendenza = 2
Ordinata all'origine −20

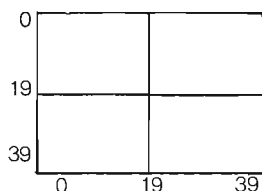
Se avessimo voluto considerare la retta Y = —X —1, non sarebbe stato possibile tracciarla con il programma precedente, in quanto, per ottenere dei valori di Y positivi, avremmo dovuto dare a X dei valori negativi. In questo caso bisogna operare un cambio di origine e considerare, ad esempio, che l'origine sia in basso a destra.

La retta tracciata ha allora come equazione:

$$39 - Y = - (X - 39) - 1 = - X + 39 - 1 = - X + 38$$

da cui Y = X + 1 considerando la nuova origine.

La traccia di curve nel modo grafico semplice

Nel migliore dei casi, la dimensione della griglia permette di avere soltanto delle curve approssimate.

Per poter essere certi di ottenere tutte le parti di curva situate nei quattro quadranti, si dovranno operare dei cambiamenti di assi. Così, considerando che il punto centrale di ascissa 19 e di ordinata 19 è la nuova origine degli assi, si può avere idea dell'andamento della curva e, dopo di ciò, operare un cambiamento di origine più appropriato.

Cambiamento di origine

Se la nuova origine ha per coordinate x_o, y_o, si sa che la nuova equazione della curva $y = f(x)$ diventa:

$$y - y_o = f(x - x_o)$$

E, poichè il senso delle y è invertito, si ha:

$$-y + y_o = f(x - x_o)$$

Esempio:

– Se si prende $x_o = 0$ e $y_o = 39$, si ottiene:

$$Y = 39 - f(x)$$

– Se si prende $x_o = 20$ e $y_o = 20$, si ottiene:

$$Y = 20 - f(x - 20)$$

Applicazione

Si debba rappresentare la parabola $y = x^2$, otterremo:

```
 10    DEF FN YP (X) = X↑2
 15    INPUT "ORIGINE"; X0, Y0
 20    INPUT "SCALA"; S
 30    GR : COLOR = 1
 40    PLOT X0, Y0
 50    REM VISUALIZZARE LA CURVA
 60    COLOR = 2
 70    FOR X = 0 TO 39
 80    A = X – X0
 90    Y = INT (Y0 – FN YP (A) * S + 0.5)
100    IF Y < 0 OR Y > 39 THEN 120
110    PLOT X, Y
120    NEXT X
130    END
```

Nota. – Il grafico semplice è poco adatto se si tratta di tracciare delle curve. Riprenderemo il problema utilizzando il grafico ad alta risoluzione.

L'interazione con un sistema grafico

Nel caso in cui si disponga di una zona al limite inferiore dello schermo che non è utilizzata per i grafici, è possibile entrare e uscire dal testo da questa parte libera, detta «finestra». Tuttavia, questa finestra non permette di punteggiare direttamente sulla parte grafica dello schermo. I meccanismi di interazione su un sistema grafico sono diversi: si possono citare la matita luminosa, il manico di scopa («joy stick»), il topo («mouse)», la matita associata ad un tavolo da disegno,... ecc.

In tutti i casi, bisogna trovare le coordinate di un punto (X, Y) con un meccanismo controllato dall'uomo. Il sistema più semplice, utilizzato specialmente dalle macchine da gioco e sul sistema «APPLE», è di disporre di manopole, associate a potenziometri (resistenze variabili) che danno i valori X e Y. Le funzioni associate sono chiamate PDL (da paddle = manopola). Così,

su un sistema grafico, si può disporre di due manopole le cui posizioni definiscono le coordinate di un punto.

Queste funzioni vengono chiamate dal programma sotto la forma PDL (0) o PDL (1) e danno un valore intero che va da 0 a 256.

Per esempio: X = PDL (0) e Y = PDL (1)

Possono esserci fino a quattro manopole.
La sintassi di questa istruzione è:

PDL (espressione intera)

ma, praticamente, il valore dell'espressione deve essere compreso tra 0 e 3.

Nel caso in cui il valore dell'espressione sia compreso tra 4 e 255 il risultato della funzione PDL è imprevedibile.

Applicazione al posizionamento di un punto sullo schermo

Nel caso in cui il valore dato dalla funzione PDL sia compreso tra 0 e 255, e che le coordinate del punto vadano da 0 a 39, bisogna dividere il valore ottenuto per un fattore

$$\frac{255}{39} = 6.$$

Dividendo dunque la funzione PDL per 6, si ottengono dei valori compresi tra

$$0 \text{ e } \frac{255}{6} = 42.$$

Basta dunque scrivere, ad esempio:

```
X  =  PDL (0)/6
IF    X > 39            THEN X = 39
Y  =  PDL (1)/6
IF    Y > 39            THEN Y = 39
```

Così si possono spaziare tutti i punti dello schermo.

Si può ugualmente scrivere:

```
X = PDL (0) * 39/255
Y = PDL (1) * 39/255
```

Esempio:

Traccia di uno schizzo sotto controllo delle manopole:

```
10    INPUT "COLOR"; C
20    GR : COLOR = C
```

```
 30   X = PDL (0) * 39 / 255
 40   Y = PDL (1) * 39 / 255
 50   IF X = X0 AND Y = Y0 THEN 30
 60   PLOT X, Y
 70   X0 = X
 80   Y0 = Y
 90   GO TO 30
100   END
```

Esercizi

1. *Scrivere un programma di gioco di «Mastermind». Il programma sceglie quattro pedine attraverso sei colori: BLU, ROSSO, VERDE, GIALLO, ARANCIONE, COLOR MALVA. Si possono avere più pedine dello stesso colore. Il giocatore indica la sua scelta, con l'aiuto delle lettere corrispondenti alla prima lettera del colore:*

   (B, R, V, G, A, M).

2. *Scrivere un programma per il gioco del quadrato cinese: ciascun giocatore sceglie un colore:*

   ```
   1   2   3
   4   5   6
   7   8   9
   ```

   Il giocatore piazza i punti del suo colore per tentare di fare una serie di tre punti, mentre l'altro giocatore glielo impedisce. Le caselle del quadrato sono numerate come nel quadrato precedente.

3. *Le torri di Hanoi.*
   Inizialmente esiste una torre composta da dischi o anelli di diametro decrescente.
   Lo scopo è di trasportare gli anelli per costituire un'altra torre, avendo solo la possibilità di deporre gli anelli di più piccolo diametro su una terza torre.
   Ad un determinato istante non si possono avere più di tre torri di anelli.

I GRAFICI AD ALTA RISOLUZIONE

Si è visto che l'utilizzo dei grafici di debole risoluzione è, in qualche modo, limitato quando si vogliono tracciare delle curve. Il grafico ad alta risoluzione utilizza una griglia di punti tale da ottenere delle tracce più precise.

Abbiamo visto che il grafico ad alta risoluzione sul sistema APPLE utilizza una griglia di punti di 280 su 160, con una finestra per il testo.

Il passaggio in modo grafico con finestra si fa con l'aiuto del comando:

HGR

mentre il passaggio in modo grafico per tutto lo schermo si fa con l'ausilio del comando:

HGR2

Il colore

La gamma dei colori è più limitata ed è specificata dal comando:

HCOLOR =
Espressione intera.

Esistono otto possibilità codificate dai numeri interi da 0 a 7:

0 = nero	4 = nero
1 = blu	5 dipende dal televisore
2 = verde	6 dipende dal televisore
3 = bianco	7 = bianco

Come si è visto, si hanno due bianchi e due neri, mentre altri colori possono variare a seconda del televisore, in funzione della tecnica utilizzata per interfacciarsi con il televisore stesso.

Visualizzazione di un punto

Ci si serve dell'istruzione HPLOT che è analoga all'istruzione grafica PLOT.

La sintassi è allora:

HPLOT espressione 1, espressione 2

dove espressione 1 ed espressione 2 rappresentano le coordinate (X e Y) dei punti da visualizzare.

Esempio:

Si debbano visualizzare le funzioni SENO e COSENO. Il programma relativo sarà:

```
10      HGR
20      FOR I = 0 TO 20 STEP 0.1
25      REM SCALA A 50 SU Y
```

```
 30    K = INT (50 * (SIN (I) + 1))
 40    J = INT (1 * 10)
 50    L = INT (50 * (COS (I) +1))
 60    HCOLOR = 2
 70    HPLOT J, K
 80    HCOLOR = 1
 90    HPLOT J, L
100    NEXT I
```

La generazione di segmenti (vettori)

Il grafico ad alta risoluzione presuppone altresì la possibilità di generare dei segmenti che uniscono due punti di coordinate (x_1, y_1) e (x_2, y_2).

Ciò potrebbe essere svolto calcolando l'equazione della retta data da:

$$\frac{x - x_1}{y - y_1} = \frac{x_2 - x_1}{y_2 - y_1}$$

In pratica, i sistemi grafici ad alta risoluzione forniscono delle facilitazioni per la generazione di vettori di questo tipo.

Così, nel sistema APPLE, l'istruzione HPLOT può essere utilizzata sotto la forma:

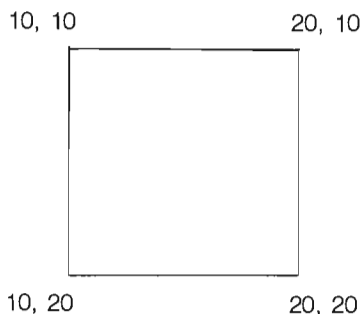
TRACCIA FINO A espressione 1, espressione 2

che indica che si vuole tracciare un vettore dal punto precedentemente visualizzato fino al punto di coordinate (x, y) specificate dalle istruzioni 1 e 2 che seguono la parola TO (FINO A).

La forma più generale dell'istruzione HPLOT permette di domandare la traccia di più vettori, l'uno di seguito all'altro:

HPLOT espr. 1, espr. 2 TO espr. 3, espr. 4...
espr. *k*, espr. *k + 1*

Esempio:

Si debba tracciare il quadrato di cui diamo i seguenti vertici:

Il programma è:

```
10      HGR
20      HCOLOR = 2
30      HPLOT 10, 10, TO 10, 20 TO 20, 20 TO 20, 10 TO 10, 10
```

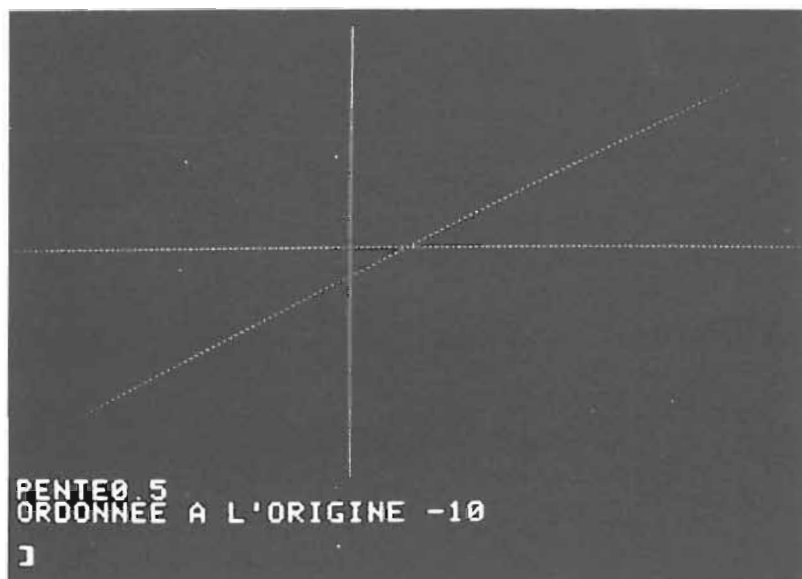
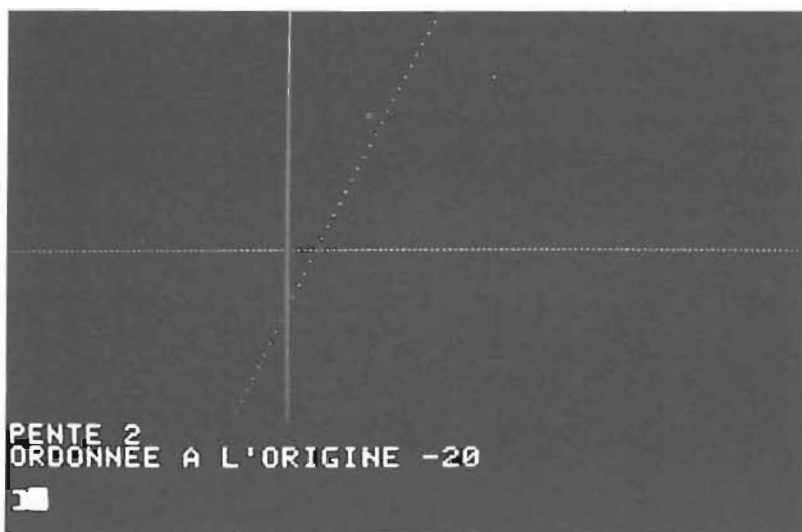
Nota. — Le espressioni corrispondenti alle ascisse devono avere dei valori compresi tra 0 e 279, mentre quelle corrispondenti alle ordinate devono essere comprese tra 0 e 159 o 191, in funzione della dimensione della griglia utilizzata.

Traccia di rette

Se si riprende il problema della traccia della retta $y = ax + b$, questa volta si ottiene una traccia molto più rigorosa.

Riprendendo il programma già scritto per il grafico di bassa risoluzione, si ha:

```
  1     REM TRACCIA DELLA RETTA
 10     INPUT "ORIGINE"; X0, Y0
 20     HGR: HCOLOR = 2
 30     REM TRACCIA DEGLI ASSI
 40     HPLOT 0, Y0 TO 279, Y0
 50     HPLOT X0, 0 TO X0, 159
 60     INPUT "PENDENZA"; A
 70     INPUT "ORDINATA ALL'ORIGINE"; B
 80     FOR X = 0 TO 279
 85     X1 = X – X0
 90     Y = Y0 – INT (A * X1 + B)
100     IF Y < 0 OR Y > 159 THEN 120
110     HPLOT X, Y
120     NEXT X
200     END
```

Esempi:

Pendenza = 5
Ordinata all'origine −10

Pendenza = 2
Ordinata all'origine −20

Traccia di poligoni regolari

Un poligono regolare è inscrivibile in un cerchio: le coordinate dei suoi vertici possono dunque essere ottenute con funzioni trigonometriche. Se il raggio del cerchio circoscritto al poligono è R, le coordinate dei vertici sono R cos A e R sen A, dove A è l'angolo sotto il quale è visto un lato del poligono, a partire dal centro. $A = 2 \pi / n$, dove n è il numero dei lati.

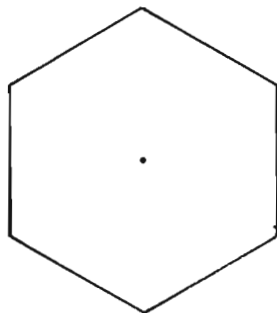
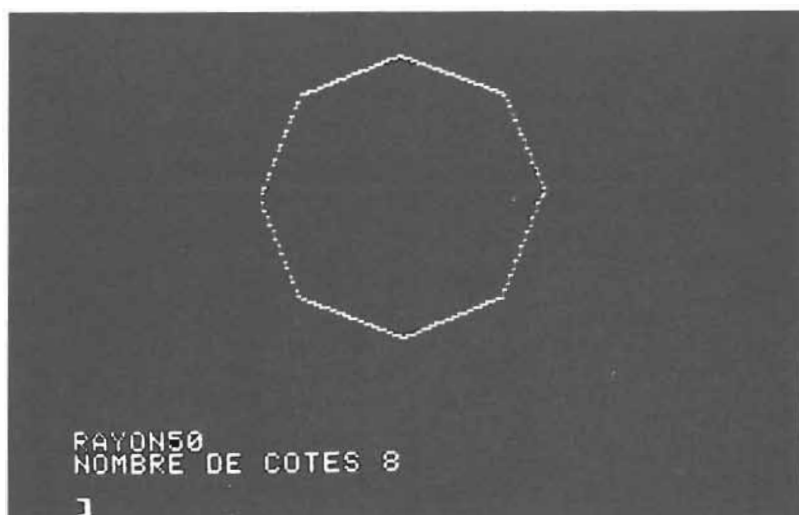
Esempio:

Per tracciare il poligono, è sufficiente unire tutti i punti così ottenuti.

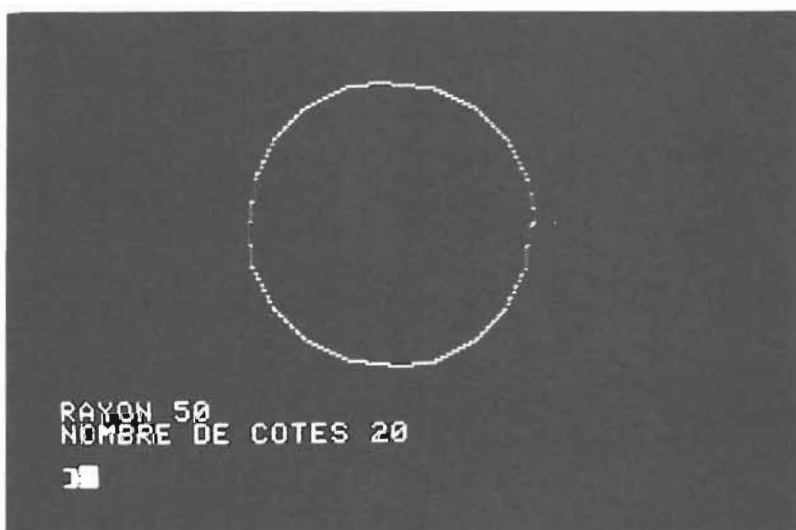
Il programma è:

```
  1     REM TRACCIA DEL POLIGONO REGOLARE
 10     INPUT "ORIGINE"; X0, Y0
 20     INPUT "RAGGIO"; R
 30     INPUT "NUMERO DI LATI"; N
 40     HGR: HCOLOR = 3
 50     PI = 3.14
 60     A = 2 * PI / N
 70     HPLOT X0 + R, Y0
 75     FOR I = 1 TO N
 80     X = X0 + R * COS (A * I)
 90     Y = Y0 – R * SEN (A * I)
110     HPLOT TO X, Y
120     NEXT I
200     END
```

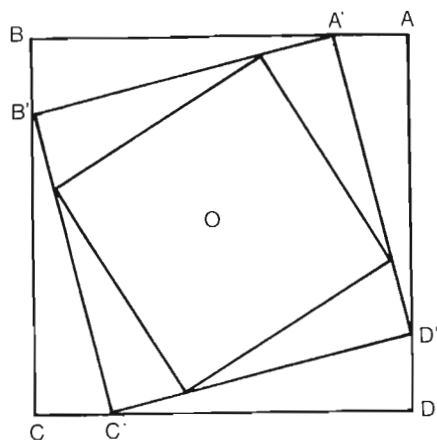
Esecuzione:

```
        ORIGINE          100,  80   ®
        RAGGIO                 20   ®
        NUMERO DI LATI          6   ®
```

Tracciato di un esagono

Raggio 50
Numero di lati 8

Applicazioni al tracciato di un cerchio

Utilizzando il programma precedente con N = 100, si ottiene una figura abbastanza simile ad un cerchio.

Raggio 50
Numero di lati 20

Esercizio:

Scrivere un programma che permetta di tracciare dei quadrati inseriti l'uno nell'altro come in figura:

Per fare ciò, è sufficiente che AA' = BB' = CC' = DD'. D'altra parte si ha:

$$OA' = OA + AA' = OB' = OB + BB' = OC' = OC + CC' = OD' = OD + DD'$$

In termini generali, se le coordinate di A sono (x, y), quelle di B sono $(y, -x)$, quelle di C = $(-x, -y)$, quelle di D = $(-y, x)$.
E:

$$AA' = \frac{AB}{K} \qquad AB = OB - OA = (y - x, \ -x - y)$$

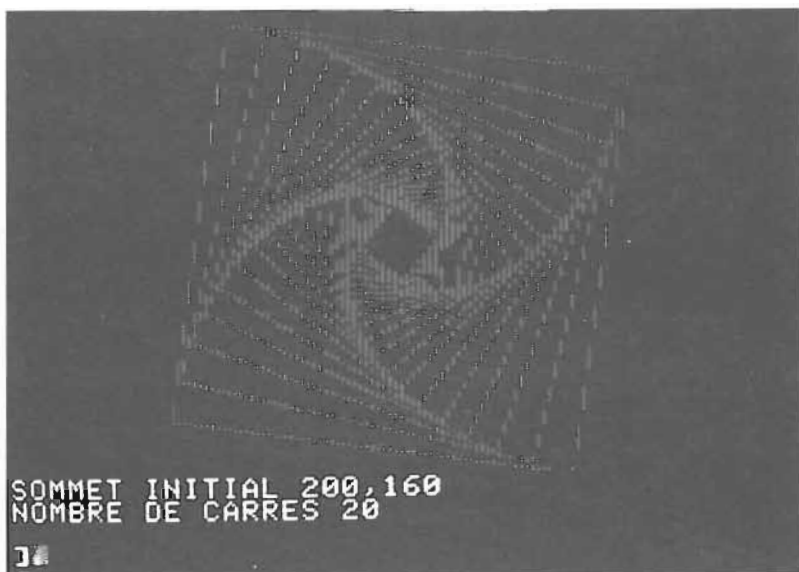
$$AA' = \frac{y - x}{K}, \ -\frac{x + y}{K}$$

Per cui:

$$\overrightarrow{OA'} = \overrightarrow{OA} + \overrightarrow{AA'} = x + \frac{y - x}{K}; y - \frac{y + x}{K}$$

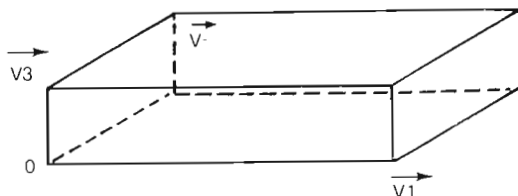
Si possono dedurre dunque le coordinate di B', C', D'.

```
 1    REM QUADRATI INCASTRATI
10    INPUT "ORIGINE"; X0, Y0
20    INPUT "VERTICE INIZIALE"; X, Y
30    INPUT "NUMERO DEI QUADRATI"; N
40    HGR: HCOLOR = 2
50    K = 10
60    X1 = X - X0
```

```
70    Y1 = Y − Y0
80    FOR I = 1 TO N
90    HPLOT X1 + X0, Y0 − Y1 TO Y1 + X0, Y0 + X1
      TO − X1 + X0, Y0 + Y1 TO − Y1 + X0, Y0 − X1
95    HPLOT TO X1 + X0, Y0 − Y1
100   X1 = X1 − (X1 − Y1) / K
120   Y1 = Y1 − (X1 + Y1) / K
130   NEXT I
```

Vertici iniziali 200, 160
Numero di quadrati 20

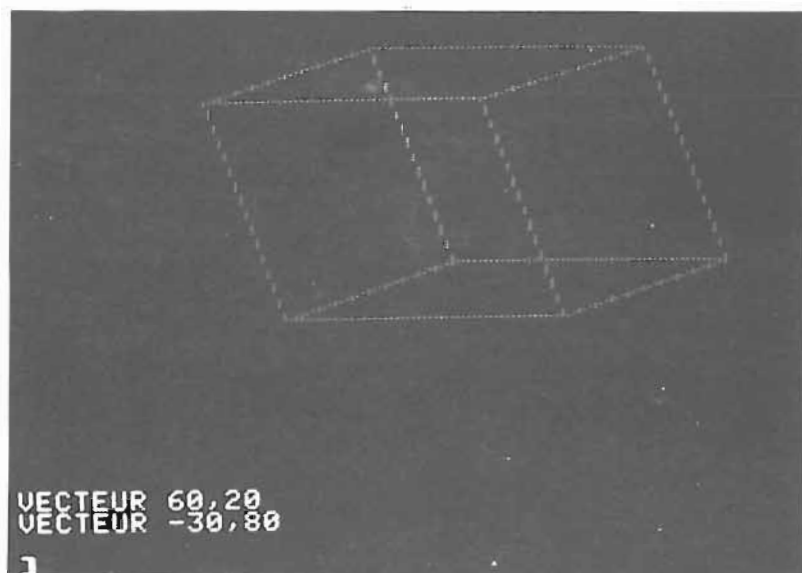
Esempio di quadrati inseriti l'uno nell'altro

Figure nello spazio

Si debba tracciare un parallelepipedo:

I vertici di un solido di questo tipo sono dati attraverso le combinazioni di tre vettori $\vec{V}1$, $\vec{V}2$, $\vec{V}3$.

Ciascun vertice è tale che:

$$OM = b_1\vec{V}_1 + b_2\vec{V}_2 + b_3\vec{V}_3$$

con $b_i = 0$ o 1

Vettore 60, 20
Vettore −30, 80

I vertici possono quindi essere rappresentati dalle terne dei valori bi. Queste terne sono rappresentate con lo schema visto qui sopra. Si può d'altra parte verificare che, se si parte dal vertice (0, 0, 0), si avranno tre direzioni possibili per tracciare un nuovo vertice, poi, partendo da (1, 0, 0) o (0, 0, 1) o (0, 1, 0), si hanno due direzioni possibili per ciascuno dei casi.

In termini del tutto generali, per un vertice (b_1, b_2, b_3) si hanno tanti spigoli che partono, quanti se ne hanno da $bi = 0$. D'altra parte, la direzione dei vettori che partono da questo vertice corrisponde al vettore V_i.

Così, per esempio:

- dal vertice 0, 0, 1 partono gli spigoli equipollenti a V_1 e V_2.
- dal vertice 1, 1, 0 parte uno spigolo equipollente a V_3

Si ricorda che un vettore $\vec{V}$ può essere caratterizzato dai suoi componenti VX e VY sugli assi delle x e delle y.

$$\vec{V} = VX \times \vec{i} + VY \times \vec{j}$$

dove $\vec{i}$ e $\vec{j}$ sono i vettori unitari.

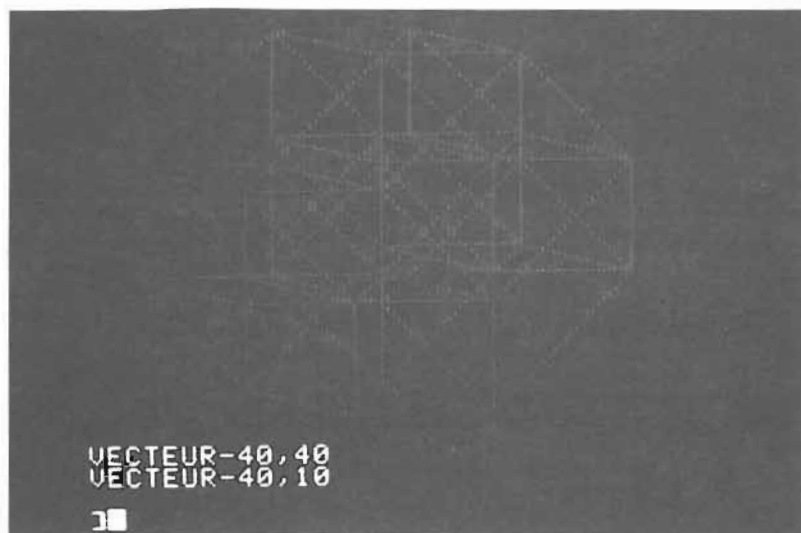
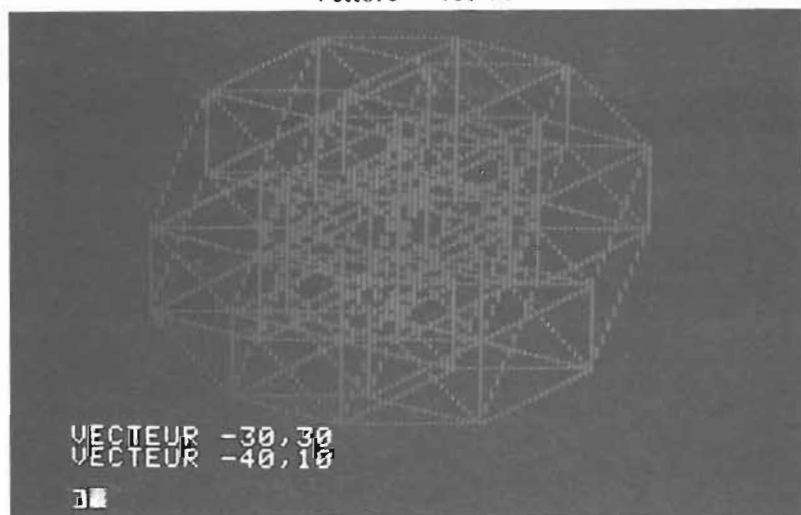
La somma dei due vettori è ottenuta aggiungendo le rispettive componenti dei due vettori.

Nota. — Questo metodo può essere generalizzato per la traccia di un iperparallelepipedo, partendo da un numero n di vettori.

Per fare ciò, è sufficiente modificare il programma precedente, introducendo una variabile N al posto delle tre. Il numero dei vertici sarà allora di 2^N.

Il programma è il seguente.

```
  1   REM TRACCIA DI UN IPERPARALLELEPIPEDO
  5   DIM X (100), Y (100)
 10   INPUT "ORIGINE"; X0, Y0
 15   INPUT "DIMENSIONI"; K
 20   HGR : HCOLOR = 2
 30   HPLOT X0, Y0
 40   FOR I = 0 TO K - 1
 50   INPUT "VETTORE"; VX (I), VY (I)
 60   B (I) = 0
 70   NEXT 1
 75   REM CONSIDERARE I 2 ↑ N VERTICI
 80   FOR I = 0 TO 2 ↑K - 1
 84   REM CONVERSIONE IN BINARIO
 85   N = I
 90   FOR J = K - 1 TO 0 STEP - 1
100   B (J) = INT (N / 2 ↑ J)
110   N = N - B (J) * 2 ↑ J
120   NEXT J
124   REM CALCOLO DELLE COORDINATE DEL VERTICE I
125   X (I) = X0 : Y (I) = Y0
130   FOR J = 0 TO K - 1
150   IF B (J) = 0 THEN 180
160   X (I) = X (I) + VX (J)
170   Y (I) = Y (I) + VY (J)
180   NEXT J
185   REM TRACCIA DEI VETTORI DERIVATI DAL VERTICE I
190   FOR J = 0 TO K - 1
200   IF B (J) < > 0 THEN 240
210   XV (J) = X (I) + VX (J)
220   YV (J) = Y (I) + VY (J)
230   HPLOT H (I), Y (I) TO XV (J), YV (J)
240   NEXT J
250   NEXT I
300   END
```

Programma che traccia un iperparallelepipedo

Vettore −40, 40
Vettore −40, 10

Vettore −30, 30
Vettore −40, 10
Figura ottenuta per un iperparallelepipedo

Tracce di curve

Nel grafico ad alta risoluzione, è possibile tracciare con buona approssimazione qualunque curva algebrica.

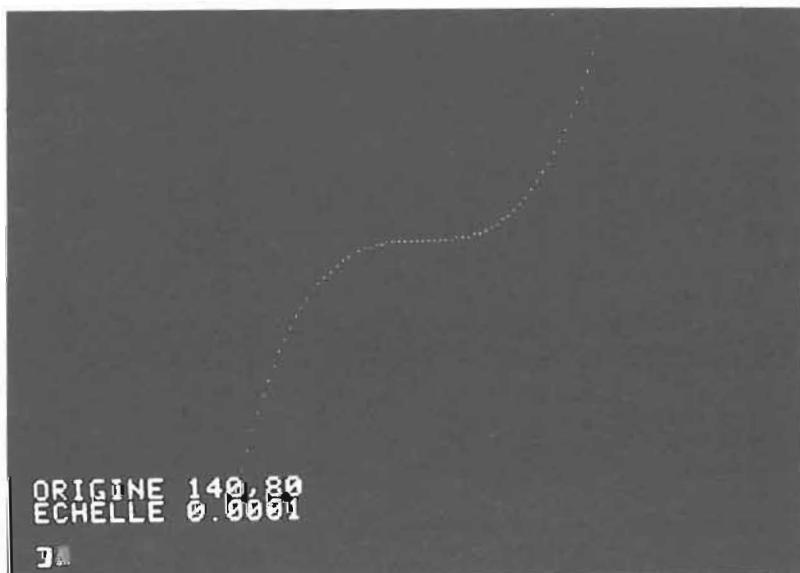
Come si è visto per il grafico a bassa risoluzione, il problema principale è quello della scala.

Quando si voglia visualizzare una curva, si comincia dunque utilizzando una scala 1 : 1, successivamente, in funzione dei risultati ottenuti, la si può modificare per ottenere una curva più adatta alle dimensioni dello schermo.

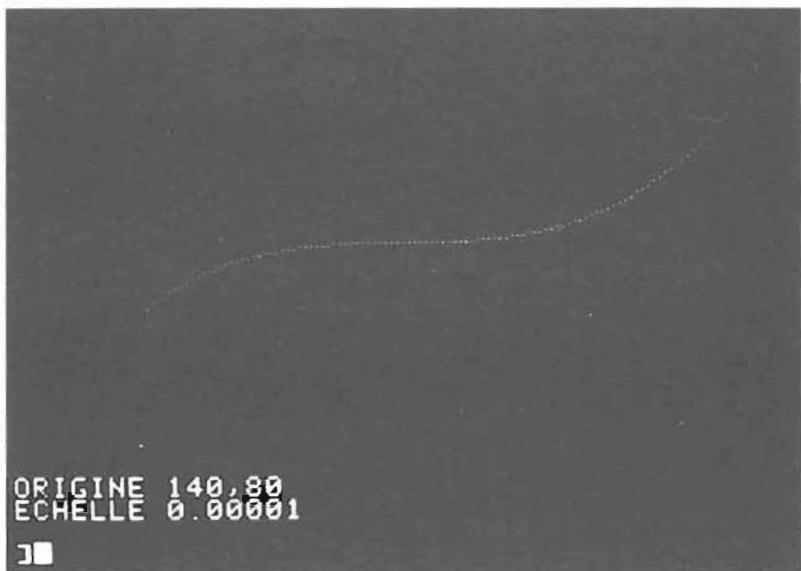
Applicazioni alla traccia di polinomi

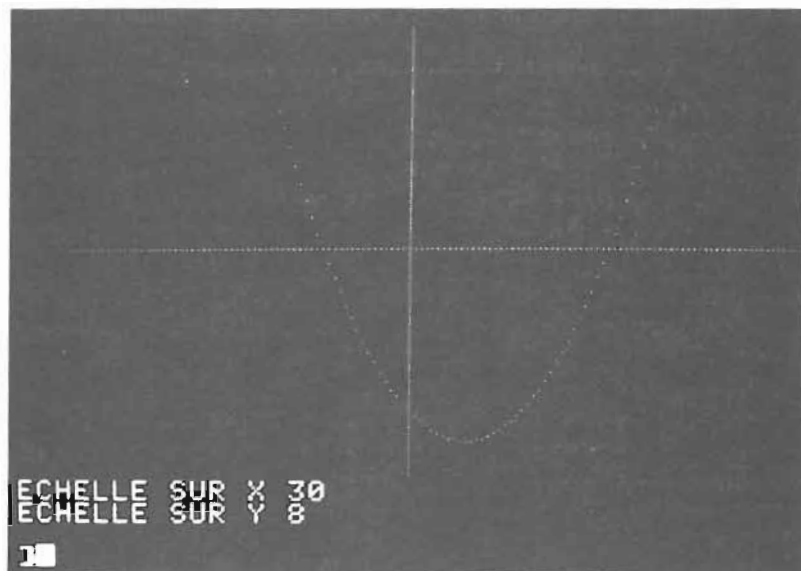
```
  1   REM TRACCIA DI POLINOMI
 10   INPUT "ORIGINE"; X0, Y0
 20   DEF FN FP (X) = X↑5 – 3 * X↑4 + 2 * X↑3 – X + 1
 30   INPUT "SCALA"; S
 40   HGR : HCOLOR = 2
 42   REM TRACCIA DEGLI ASSI
 43   HPLOT 0, Y0 TO 279, Y0
 44   HPLOT X0, 0 TO X0, 159
 50   FOR X = 0 TO 279
 60   A = X – X0
 70   Y = INT (Y0 – FN FP (A) * S + 0.5)
 80   IF Y < 0 OR Y > 159 THEN 100
 90   HPLOT X, Y
100   NEXT X
200   END
```

Esecuzione:

```
ORIGINE 140,80  ®
SCALA   0.0001  ®
```

Origine 140, 80
Scala 0.0001

Introducendo poi un fattore di scala sull'asse delle Y, si ottiene il programma:

```
1     REM TRACCIA DI POLINOMI
10    INPUT "ORIGINE"; X0, Y0
20    DEF FN FP (X) = 3 * X↑2 – 4 * X – 7
30    INPUT "SCALA SULLE X"; K
35    INPUT "SCALA SULLE Y"; S
40    HGR : HCOLOR = 2
42    REM TRACCIA DEGLI ASSI
43    HPLOT 0, Y0 TO 279, Y0
44    HPLOT X0, 0 TO X0, 159
50    FOR X = 0 TO 279
60    A = X – X0
65    A = A/K
70    Y = INT (Y0 – FN FP (A) * S + 0.5)
80    IF Y < 0 OR Y > 159 THEN 100
90    NEXT X
200   END
```

Origine 140, 80
Scala 0,00001

Esempio

Scala su X 30
Scala su Y 8

Problema delle funzioni che assumono dei valori infiniti per dei valori finiti

Il programma precedente non può essere utilizzato per tracciare il grafico della funzione $y = 1 / x$.

Infatti per $x = 0$ si ha $y =$

La diramazione nel caso di errori

Per rimediare a questa difficoltà, esiste spesso una possibilità che permette di saltare ad un trattamento specifico quando si ha un errore di esecuzione. Questa istruzione è un salto condizionale su un errore. La sintassi è:

ONERR GO TO *n*
(*SU ERRORE VAI A n*)

n è il numero dell'istruzione nella quale si deve saltare, se si produce un errore.

Il tipo di errore è specificato da un codice che può essere letto grazie ad una istruzione PEEK (vedi in seguito) ad un indirizzo specifico.

Questa istruzione deve essere eseguita prima che l'errore si verifichi. Così, se si sa che una funzione assume valori infiniti a causa di divisioni per zero, è sufficiente inserire tale istruzione.

Il proseguimento dopo l'errore

Se si vuole riprendere l'istruzione sulla quale l'errore si è prodotto, si utilizzerà l'istruzione RESUME (RIPRESA).

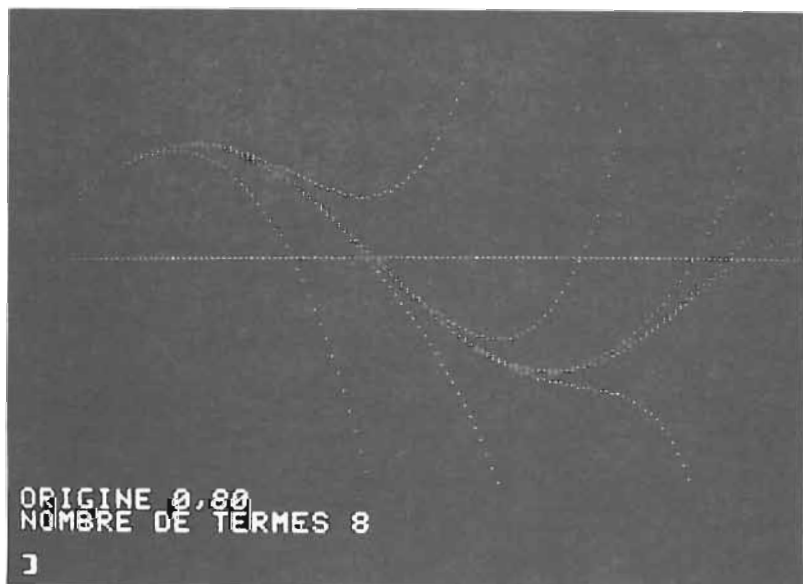
Applicazione

Ciò può essere utilizzato per tracciare funzioni razionali del tipo:

$$y = \frac{ax^2 + bx + c}{a'x^2 + b'x + c'}.$$

La modifica del programma per tracciare le curve già visto è lasciata al lettore a titolo d'esercizio.

Sviluppi limitati di funzioni

Alcuni tipi di funzioni di variabili reali possono essere approssimati per mezzo di serie polinominali.

Origine 0,80
Numero di termini 8

Per esempio:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!} + \dots$$

$$\frac{1}{1 + x} = 1 - x + x^2 - x^3 + x^4 + \dots \text{si } 0 < x < 1$$

$$\sin x = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} + \dots + (-1)^{n-1} \frac{x^{2n+1}}{(2_n + 1)!}$$

$$\text{Log}\,(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \frac{x^5}{5} + \dots \text{per } x < 1$$

È interessante verificare graficamente che questi polinomi danno delle curve sempre più ravvicinate della funzione che essi approssimano.

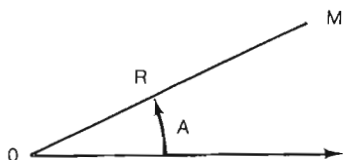
Così, per esempio, per sin x si ha il programma seguente:

```
  1   REM SVILUPPI LIMITATI
 10   INPUT "ORIGINE"; X0, Y0
 20   INPUT "NUMERO DI TERMINI"; N
 40   HGR : HCOLOR = 2
 42   REM TRACCIA DEGLI ASSI
 43   HPLOT 0, Y0 TO 279, Y0
 44   HPLOT X0, 0 TO X0, 159
 50   FOR X = 0 TO 7 STEP 0,025
 60   Y = O : F = 1
 70   HPLOT X - X0, Y0 - Y
 80   K = 1
 90   FOR I = 0 TO N
100   DL = K * X↑(2 * I + 1) / F
110   K = - K
120   F = F * 2 * (I + 1) * (2 * I + 3)
125   A = X * 40
130   Y = Y + DL * 40
135   IF Y0 - Y < 0 OR Y0 - Y > 159 THEN 150
140   HPLOT A - X0, Y0 - Y
150   NEXT I
160   NEXT X
200   END
```

Le tracce in coordinate polari

Alcune curve si esprimono molto più semplicemente utilizzando coordinate polari, in cui viene data la distanza R = OM dall'origine in funzione del-

l'angolo A formato con l'asse di origine.

R è chiamato raggio vettore e A angolo polare.

Una curva in coordinate polari è dunque espressa da una funzione del tipo:

$$R = f(A)$$

Esempi:

- Un cerchio avente il centro nell'origine è espresso in coordinate polari da R = Costante.
- R = K (1 − cos A) è una curva chiamata cardioide.
- R = KA è una curva chiamata spirale di Archimede.
- R = K sen 2 A è un rosone a quattro petali.

Il passaggio dalle coordinate polari alle coordinate cartesiane è estremamente semplice, poiché il punto M ha coordinate x e y tali che:

$$x = R \cos A$$
$$y = R \sin A$$

Diventa così possibile dare le curve in coordinate polari e tracciarle utilizzando l'istruzione HPLOT.

Si debba tracciare la curva R = K (1 − cos A); avremo il seguente programma:

```
 1   REM TRACCIA DI CARDIOIDE
10   INPUT "ORIGINE"; X0, Y0
20   INPUT "PARAMETRO"; K
25   HGR : HCOLOR = 3
26   PI = 3.14
30   FOR A = 0 TO 2 * PI STEP PI / 20
40   X = K * (1 − COS (A)) * COS (A)
50   Y = K * (1 − COS (A)) * SIN (A)
60   HPLOT X + X0, Y0 − Y
70   NEXT A
80   END
```

In questo modo si ottiene la curva il cui grafico è dato qui sotto:

Origine 140, 80
Parametro 60

Traccia di una cardioide

Tracce di rosoni e di «fiori»

Mostriamo qui la famiglia di curve la cui equazione polare è

$$R = K + \cos(n\ A)$$

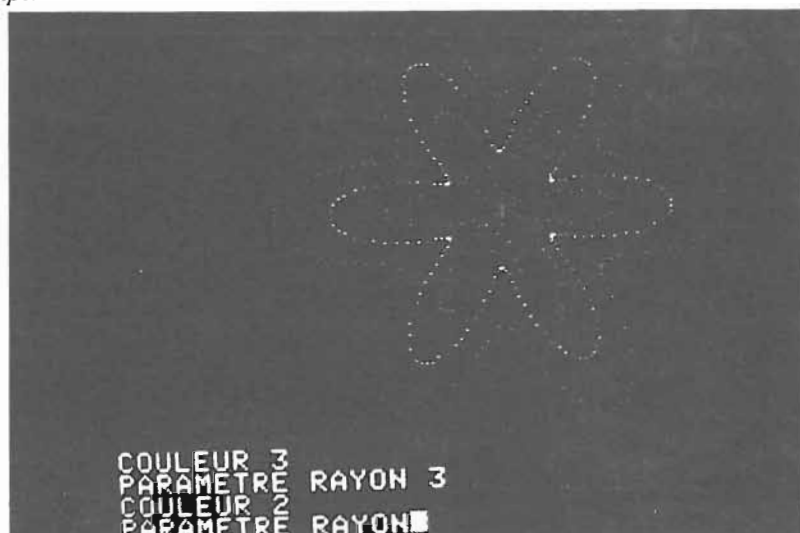
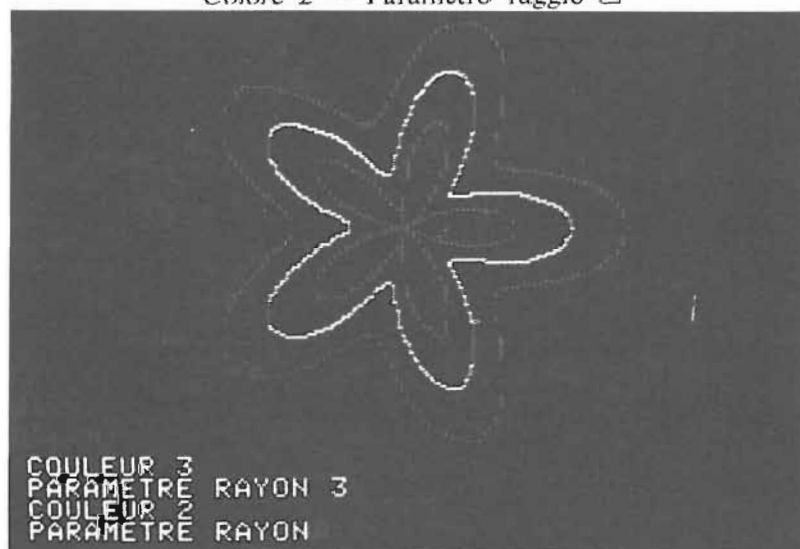
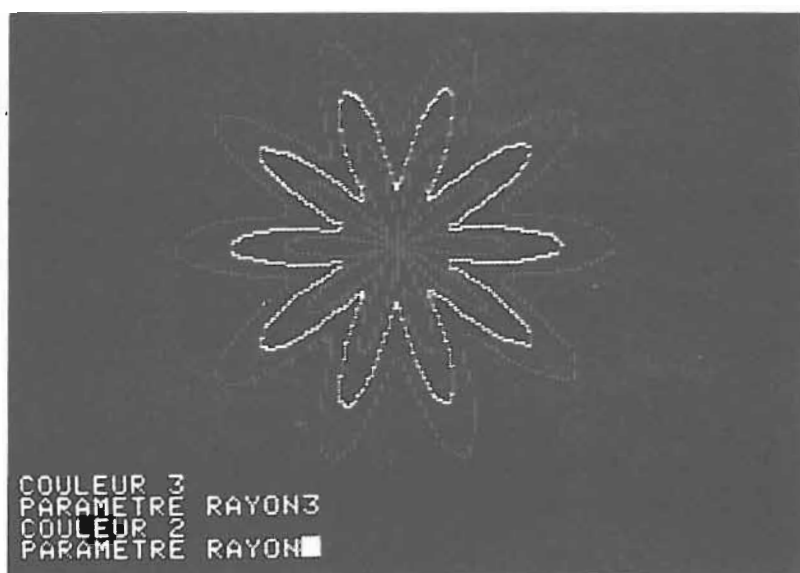
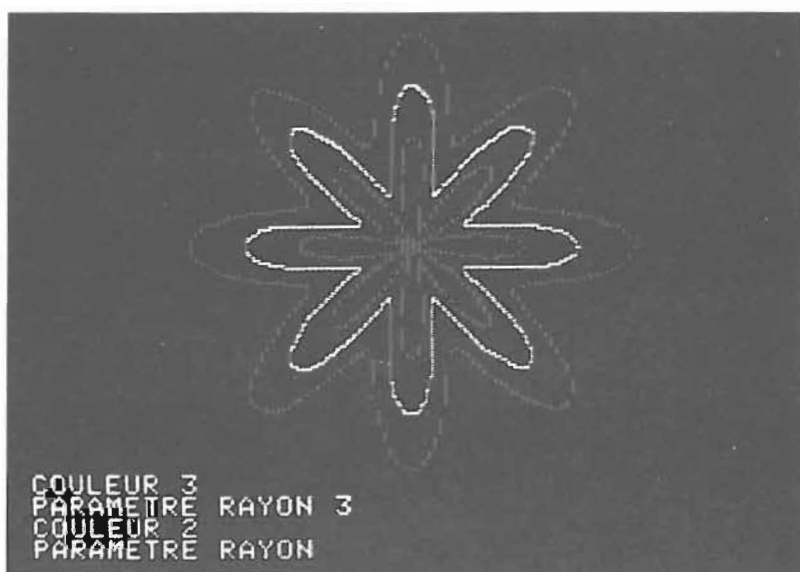
Essa rappresenta le proiezioni della traiettoria di un punto che si muove con regolarità sulla superficie di una forma toroidale.

Le curve corrispondenti hanno la forma di rosoni o di fiori.

Il programma corrispondente è:

```
 1    REM TRACCIA DI ROSONI
10    INPUT "ORIGINE"; X0, Y0
20    INPUT "PARAMETRO ANGOLARE"; N
26    PI = 3.14
30    HGR
40    INPUT "PARAMETRO RAGGIO"; K
50    INPUT "COLORE"; C
55    HCOLOR = C
56    HPLOT X0 + 20 * K + 20, Y0
60    FOR A = 0 TO 2 * PI STEP PI / 100
70    X = (K + COS (N * A)) * COS (A)
80    Y = (K + COS (N * A)) * SIN (A)
85    X = X * 20
86    Y = Y * 20
```

```
90    HPLOT TO X + X0, Y0 – Y
100   NEXT A
110   GO TO 40
200   END
```

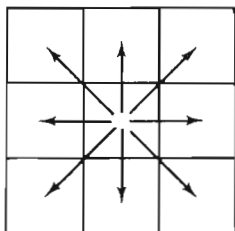
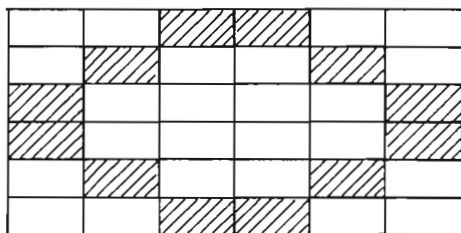
Esempi:

Colore 3 – Parametro raggio 3
Colore 2 – Parametro raggio □

Colore 3 – Parametro Raggio 3
Colore 2 – Parametro raggio □

Colore 3 – Parametro raggio 3
Colore 2 – Parametro raggio □

Colore 3 – Parametro raggio 3
Colore 2 – Parametro raggio □

La definizione di forme grafiche

Negli esempi precedenti abbiamo trattato delle curve che possono esprimersi attraverso equazioni algebriche. Abbiamo visto anche come trattare delle immagini che possono essere costruite in modo algoritmico.

Tuttavia, volendo creare forme definite punto per punto, e poi lavorare su queste forme in modo algoritmico, dovremo definire dei meccanismi che permettono di definirle e di trattarle.

Qualunque sia il sistema utilizzato, una forma è un insieme di punti e, se si dispone di una istruzione che permette di visualizzarla punto per punto, è interessante definire una forma con un processo continuo.

In una griglia di punti, il posizionamento può farsi in otto direzioni date qua sotto.

Si debba definire la forma seguente:

Se si parte dall'alto e nel senso delle lancette dell'orologio, avremo dunque i seguenti movimenti:

→ ↘ ↘ ↓ ↙ ↙ ← ↖ ↖ ↑ ↗ ↗

A questi movimenti può essere attribuito un codice, ad esempio:

→ D (destra)
← S (sinistra)
↑ A (alto)
↓ B (basso)
↗ DA (destra alto)
↘ DB (destra basso)
↖ SA (sinistra alto)
↙ SB (sinistra basso)

La sequenza precedente può essere allora così codificata: D, DB, DB, B, SB, SB, S, SA, SA, A, DA, DA

Programma di creazione e visualizzazione di una forma

Utilizzando il sistema di codifica precedente, ci si propone di creare e visualizzare una forma. Aggiungiamo due lettere A per annullare il movimento precedente e T per terminare.

Il programma seguente permette di creare e visualizzare una forma:

```
 10   DIM F$ (100)
 20   INPUT "ORIGINE"; X0, Y0
 30   HGR : HCOLOR = 2
 40   HPLOT X0, Y0
 50   X = X0 : Y = Y0
 55   FOR I = 1 TO 1000
 60   INPUT F$ (I)
 70   IF F$ (I) = "A" THEN HCOLOR = 0 : HPLOT, X,
      Y : GO TO 60
 80   IF F$ (I) = "T" THEN 220
 90   HCOLOR = 2
100   IF F$ (I) = "D" THEN X = X + 1 : GO TO 200
110   IF F$ (I) = "S" THEN X = X - 1 : GO TO 200
120   IF F$ (I) = "A" THEN Y = Y - 1 : GO TO 200
130   IF F$ (I) = "B" THEN Y = Y + 1 : GO TO 200
140   IF F$ (I) = "SB" THEN X=X-1:Y=Y+1:GOTO200
150   IF F$ (I) = "DB" THEN X=X+1:Y=Y+1:GOTO200
160   IF F$ (I) = "DA" THEN X=X+1:Y=Y-1:GOTO200
170   IF F$ (I) = "SA" THEN X=X-1:Y=Y-1:GOTO200
175   GO TO 60
176   REM
180   REM VISUALIZZARE IL PUNTO X, Y
190   REM
200   HCOLOR = 2 : HPLOT X, Y
210   NEXT I
220   END
```

Codice interno della forma

A partire dalla tabella F$, è possibile creare una tabella di codici interni, bastano infatti tre bits per rappresentare l'insieme degli otto spostamenti possibili.

Nel sistema APPLE il sistema di codifica degli spostamenti è sensibilmen-

te differente. In effetti, si suppone che ci si limiti a spostamenti ad angolo retto. Se il punto è visualizzato, lo si indica con l'aiuto di una freccia con un punto.

Il sistema di codifica utilizzato è:

↑ 000	↓ 100
→ 001	← 101
↓ 010	↑ 110
← 011	→ 111

È facile passare dal sistema di codifica precedente a questo, tenendo conto che gli spostamenti obliqui sono l'equivalente di due spostamenti ortogonali:

DA = ↗ = → ↑

DB = ↘ = → ↓

SA = ↖ = ← ↑

SB = ↙ = ← ↓

È altresì possibile produrre una forma codificata in una tabella che si chiama tabella di forma.

Nel sistema APPLE si raggruppano due o tre codici in un ottetto, a seconda che si abbiano degli spostamenti con o senza visualizzazione. La tabella delle forme ne può contenere diversi ed è disposta ad un indirizzo preciso di memoria.

Le istruzioni di manipolazione della forma

Esiste una istruzione che permette di tracciare una forma partendo da una tabella in memoria.

TRACCIA espressione 1	A espressione 2, espressione 3
(DRAW)	(AT)

L'espressione 1 specifica il numero d'ordine della forma nella tabella.

Le espressioni 2 e 3 specificano le coordinate dell'origine. Esiste una seconda versione che permette di fare una traccia di colore complementare a ciascun punto visualizzato. È l'istruzione C TRACCIA o XDRAW. I colori complementari sono Nero e Bianco o Blu e Verde. L'interesse di questa istruzione sta nel fatto che due istruzioni XDRAW successive permettono di cancellare una forma senza modificare tutto ciò che sta intorno.

Le altre istruzioni sono istruzioni di cambio di scala e di rotazione della forma.

La messa in scala 0 o il cambio di scala vengono effettuate con l'istruzione:

SCALE = espressione

L'espressione è un valore intero che precisa la scala da utilizzare (compreso tra 0 e 255).

La rotazione della forma si effettua con l'istruzione:

ROT = espressione

Il valore dell'espressione può variare da 0 a 255, ma, per la scala di 1, solo i valori 0, 16, 32, 48 sono riconosciuti e permettono di fare delle rotazioni di 90°, 180°, o 270° nel senso di rotazione delle lancette dell'orologio. Per una scala di 2 si hanno otto rotazioni possibili..., ecc.

Quando un valore non è riconosciuto, viene utilizzato come valore corretto quello ad esso più prossimo.

Esempio:

```
10        HGR
20        H COLOR = 2
30        FOR I = 1 TO 64
40        ROT = I
50        SCALE = I
60        DRAW 1 AT 100, 80
70        NEXT I
```

Questo programma permette di tracciare una forma, partendo dal punto di coordinate (100, 80), facendo variare la scala e facendo ruotare la forma.

CONCLUSIONE

Questo capitolo è senza dubbio il più originale in un'opera di introduzione al BASIC.

Effettivamente, la grafica è spesso considerata come un lusso o come un «gadget» inutile all'apprendimento della programmazione.

Nel primo caso, si pensa a delle periferiche estremamente costose, e, nel secondo caso, si pensa a dei procedimenti semigrafici per i giochi-video.

Lo sviluppo dei sistemi grafici su televisori standard ha permesso di mostrare che ciò non è più un lusso e che i procedimenti grafici possono essere utilizzati per dei programmi seri, oltre che per i giochi.

In particolare, i procedimenti grafici permettono di presentare i risultati in maniera più leggibile di quanto lo siano le tabelle di cifre.

Infine, il procedimento grafico possiede una qualità essenziale che non può essere sostituita da nessun altro mezzo di uscita: si tratta della rappresentazione del tempo sotto forma di movimento. Ciò può essere utilizzato non solo per dei giochi, ma pure per la rappresentazione dinamica dei risultati, la simulazione dei processi dinamici, l'animazione e la rappresentazione di forme artistiche o geometriche.

CAPITOLO 7

LE ISTRUZIONI SPECIFICHE DI ALCUNI SISTEMI BASIC

Il BASIC è senza dubbio il linguaggio che possiede il maggior numero di varianti nelle estensioni che offrono i differenti sistemi. In questo capitolo, non pretendiamo certo di elencare tutte le varianti. Ci limiteremo alle istruzioni che sono presenti in diversi sistemi e che hanno una certa utilità.

LE ISTRUZIONI «SISTEMA»

Intendiamo definire con questo termine alcune istruzioni che permettono di accedere alla memoria della macchina o di specificare il salto ad un sottoprogramma in linguaggio macchina.

Lettura di una memoria

È l'istruzione:

PEEK (indirizzo di memoria)

L'indirizzo memoria può essere specificato da una espressione intera che varia da 0 alla massima dimensione della memoria della macchina.

L'utilità di questa istruzione sta nel fatto che permette di leggere alcune posizioni memoria aventi delle funzioni specifiche su una macchina data (entrata - uscita).

Il valore letto è rimandato in decimale (da 0 a 255) e deve essere memorizzato in una variabile:

X = PEEK (AD)

APPLICAZIONI:

Creazione di un suono

Su alcune macchine che dispongono di uno speaker, questo può essere utilizzato con l'aiuto di questa istruzione.

Se S è l'indirizzo memoria associato a questo speaker, si può produrre un suono, scrivendo l'istruzione:

SUONO = PEEK (S)

Si può anche creare un suono più lungo scrivendo:

RUMORE = PEEK (S) − PEEK (S) + PEEK (S) − PEEK (S)

Posizione del cursore

È in generale possibile avere la posizione orizzontale e verticale del cursore con l'aiuto dell'istruzione:

CH = PEEK (H)
CV = PEEK (V)

H e V sono degli indirizzi (per esempio nel sistema APPLE 36 e 37).

Lettura di un carattere alla tastiera

Se C è l'indirizzo associato alla tastiera, l'istruzione:

X = PEEK (C)

permette di ottenere il codice ASCII del tasto che è stato premuto (con il bit 7 posizionato). In particolare, se X > 127, si sa che è stato premuto un tasto.

Scrittura in una memoria

È l'istruzione inversa della precedente. La sua sintassi è:

POKE indirizzo previsto, valore

Questa istruzione serve a mettere un valore nella memoria di cui viene specificato l'indirizzo.

ATTENZIONE! Se questa istruzione è utilizzata a torto o in maniera difettosa, può distruggere parti di programma o di dati. Bisogna quindi usarla con prudenza.

Esempi:

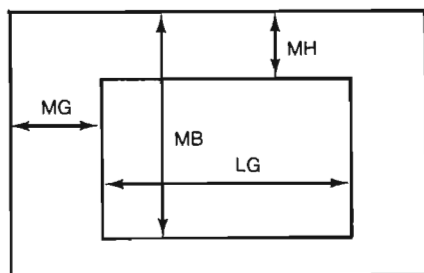
Per piazzare il cursore in una certa posizione, si possono utilizzare delle istruzioni POKE agli indirizzi sopra definiti.

POKE H, CH
POKE V, CV

Allo stesso modo, è possibile cambiare la finestra del testo sullo schermo, definendone la larghezza (da 1 a 40) ed un margine a sinistra sia in alto che in basso.

Così, le istruzioni seguenti definiscono:

POKE F, LG	la larghezza della finestra
POKE S, MS	il margine sinistro
POKE A, MA	il margine in alto
POKE B, MB	il margine in basso

(S = 32, F = 33, A = 34, B = 35 sul sistema APPLE)

Esempio:

```
10      LG = 20 : MA = 10 : MB = 30 : MS = 10
20      S = 32 : F = 33 : A = 34 : B = 35
30      POKE F, LG
40      POKE S, MS
50      POKE B, MB
60      POKE A, MA
```

Altre istruzioni POKE permettono di modificare alcuni parametri del sistema grafico o i controlli dei giochi, ecc.

Nota. – Le istruzioni PEEK e POKE esistono in forme analoghe nella maggior parte dei microelaboratori individuali. Al contrario, nei sistemi più grossi, sono assenti, in quanto dannose.

Definizione della memoria utilizzabile da un sistema BASIC

Esistono delle istruzioni o dei comandi che permettono di fissare il limite inferiore e superiore della memoria utilizzabile in un programma BASIC.

Sono le istruzioni:

HIMEM: indirizzo (la memoria più alta)
LOMEM: indirizzo (la memoria più bassa)

Gli indirizzi possono essere specificati con espressioni intere che corrispondono ad indirizzi esistenti. Se il sistema possiede solo 16 K, questo valore sarà compreso tra 0 e 16384, se invece possiede 64 K, non si potrà andare oltre 65535 (1 K = 2^{10} = 1024).

Si suppone che il valore definito da LOMEM sia inferiore a quello definito da HIMEM.

Se queste istruzioni non sono specificate, il sistema utilizza dei limiti che corrispondono a tutta la memoria disponibile per l'utente. L'effetto di queste istruzioni è annullato da una operazione di rimessa a zero (RESET o CONTROLE B).

L'istruzione di attesa

Questa istruzione permette di realizzare un ritardo programmato o una pausa. Questa istruzione permette di arrestarsi all'interno di un programma, fino al momento in cui una certa condizione risulta verificata.

La sintassi di questa istruzione è

WAIT indirizzo, espressione 1, espressione 2
(ATTENDI)

Il primo parametro è l'indirizzo di una posizione memoria che verrà valutata durante l'attesa. Questo indirizzo può essere una espressione, a condizione che ciò corrisponda ad un indirizzo valido.

Le espressioni 1 e 2 rappresentano variabili intere (da 0 a 255). L'espressione 2 è facoltativa. Se la seconda espressione è assente, l'istruzione realizza una operazione logica E (AND) tra il contenuto dell'indirizzo e il valore binario dell'espressione (bit a bit).

Se il risultato è 0, l'attesa continua, altrimenti, se il risultato dell'operazione E (AND) è diverso da 0, l'attesa termina.

Esempio:

WAIT AD, 3

Il numero 3 corrisponde a 00000011 in binario; questa istruzione provocherà dunque un ritardo fino a quando almeno uno dei due bits di destra sarà presente nella memoria di indirizzo AD. Se l'espressione 2 è presente, avremo innanzitutto una operazione O (OR) esclusivo tra il contenuto della memoria

e il valore dell'espressione 2. L'operazione OR esclusivo è quella per cui, se due bits sono identici, il risultato è nullo, altrimenti il risultato è 1.

Successivamente, il risultato ottenuto sarà usato per una operazione E (AND) con il valore dell'espressione 1.

Ciò permette di verificare la condizione per cui un bit della posizione memoria sia nullo.

Esempi:

– WAIT AD, 1, 1 specifica una pausa fino a quando il bit a destra di AD sia 0;
– WAIT AD, 3, 2 specifica una pausa fino a quando il bit a destra di AD sia uguale a 1, oppure che il secondo bit abbia un valore uguale a 0.

La chiamata di un sottoprogramma in linguaggio macchina

Questa istruzione è utile quando si vuole chiamare un sottoprogramma scritto direttamente in linguaggio macchina.

La sintassi dell'istruzione è:

CALL indirizzo
(CHIAMA)

L'indirizzo deve essere un numero intero corrispondente ad un indirizzo esistente sulla macchina (se il sistema ha 64 K si può andare fino a 65535). Questo indirizzo può essere rappresentato da una espressione valutata sotto forma di un valore intero.

L'istruzione è utile in quanto permette di far riferimento e di eseguire programmi scritti in linguaggio macchina.

Alcuni programmi sono forniti dal costruttore, specialmente quelli per cancellare lo schermo o una linea, ecc.

D'altra parte, ciò permette all'utente di scrivere programmi più specifici e di richiamarli partendo da un programma BASIC.

Il passaggio di parametri

L'istruzione precedente permette di fare un salto ad un sottoprogramma, ma spesso è necessario associare dei parametri o dei valori definiti nel programma principale.

Ciò è possibile grazie ad una istruzione del tipo (sistema APPLE):

USR (espressione)

L'espressione definisce un parametro che può essere intero o fluttuante (ma reale) e che sarà trasmesso al sottoprogramma con l'intermediazione di

un accumulatore fluttuante che generalmente si trova nella prima pagina della memoria (indirizzi da 0 a 255). Il modo dettagliato di impiego di questa istruzione è dato dai costruttori negli opuscoli.

LE ISTRUZIONI PER LA STAMPA

Quando un sistema BASIC dispone di una unità per la visualizzazione (televisione o monitor TV), è generalmente possibile definire delle istruzioni specifiche per la stampa.

Movimento del cursore

Soventemente esistono istruzioni che permettono di posizionare il cursore in un punto preciso dello schermo.

Così, HTAB e VTAB permettono di posizionare il cursore orizzontalmente e verticalmente sullo schermo. La sintassi è:

HTAB X
VTAB Y

dove X e Y sono valori interi. Per esempio, se lo schermo dispone di 24 linee di 40 caratteri, si deve avere:

$$1 \leq X \leq 24 \quad \text{e} \quad 1 \leq Y \leq 40$$

Cancellazione dello schermo

Generalmente esiste una istruzione che permette di cancellare lo schermo e di posizionare il cursore in alto a sinistra.

Questa istruzione è il comando HOME («torna alla tua posizione iniziale»). In alcuni casi, lo stesso scopo può essere ottenuto stampando un carattere speciale. Ad esempio, nel CBM (PET)

PRINT «♡»

avrà lo stesso effetto.

Lettura della posizione del cursore

Questa istruzione permette di conoscere la posizione del cursore sulla linea corrente. Si tratta dell'istruzione POS (espressione) che valuta la posizione del cursore in funzione del margine sinistro. La prima posizione corrisponde al valore 0. L'espressione può essere qualunque, ma non ha significato per POS.

Volendo conservare questa posizione, si dovrà utilizzare una istruzione del tipo:

X = POS (0)

Esempio:

```
PRINT TAB (15)
X = POS (0)          dà X = 14
PRINT X
```

Stampa di spazi

Benchè sia possibile stampare dei caratteri bianchi, è preferibile, per ottenere degli spazi, disporre di una istruzione specifica. Così l'istruzione SPC (X) (SPACE = SPAZIO) permette di stampare X spazi.

Se l'istruzione TAB parte dall'inizio di una linea, l'istruzione SPC permette di stampare X spazi, partendo dalla posizione corrente del cursore. Se si supera il bordo destro della linea, si continua sulla linea successiva.

La sintassi è:

SPC (espressione)

L'espressione è convertita in un intero tra 0 e 255. SPC (0) non dà nessun effetto (0 spazi).

Tuttavia, è possibile realizzare la stampa di un qualunque numero di spazi, aggiungendo più istruzioni SPC.

Esempio:

```
PRINT SPC (255), SPC (100)
```

permette di stampare 355 spazi bianchi.

I comandi video o televisione

Nei sistemi che utilizzano in uscita degli apparecchi TV standard, esistono comandi del tipo:

FLASH

che permettono di far variare velocemente e con continuità il colore del fondo dello schermo (da scrittura bianca su nero a scrittura nera su bianco).

Il comando INVERSE permette di passare da un testo visualizzato in nero su bianco, allo stesso testo visualizzato bianco su nero. Il passaggio al testo in bianco su fondo nero viene effettuato specificando il comando NORMAL.

La velocità con cui il testo si muove sullo schermo viene specificata dal comando SPEED.

SPEED = espressione

L'espressione può assumere dei valori da 0 a 255.

La disponibilità di memoria (FRE)

Nella maggior parte dei sistemi, esiste un comando o una istruzione che permette di conoscere lo spazio di memoria disponibile per l'utente, al di fuori dello spazio occupato dal suo programma.

Si tratta dell'istruzione FRE (espressione). In pratica, l'espressione che segue l'istruzione FRE può essere il valore 0. Il risultato è un numero intero positivo o negativo. Nel caso sia negativo, si tratterà di un valore superiore a 32767, che si ottiene aggiungendo 65536 (2^{16}).

Esempio:

X = FRE (0) se X > 0 X è il numero di parole.
X < 0 si aggiunge 65536.

L'istruzione di stampa con formato (PRINT ..... USING)

Questa istruzione non è sempre disponibile sui sistemi più piccoli, ma su sistemi di una certa dimensione la si trova abbastanza spesso.

L'istruzione è specificata da una clausola di formato introdotta dalla parola chiave USING (UTILIZZANDO).

Questa clausola permette di controllare il formato della stampa, specificando una catena di caratteri che ha un significato ben preciso e dà la forma della stampa che si vuole ottenere. In questa serie di caratteri, si indica la posizione dei segni, dei numeri, dei punti decimali, delle virgole e dei caratteri accessori.

La sintassi di questa istruzione può variare leggermente da un sistema all'altro.

Nella maggior parte dei casi, può far riferimento a un numero di formato, ma si può ugualmente specificare il formato con una istruzione PRINT.

Esempi:

– PRINT USING "# # # . # #", K permette di stampare K sotto forma di un numero di tre cifre (o di tre caratteri bianchi) prima del punto, seguito da due decimali.

Si può ugualmente avere una forma del tipo

PRINT USING F$; X, Y, Z

associata ad una istruzione di formato definita da:
F$ = "I RISULTATI X, Y, Z SONO: $$ # # . #, $$ # # . #, $$ # # . #"

Infine, si può anche scrivere:

PRINT USING 40, A, B

Questa volta il formato fa riferimento all'istruzione 40, che dà il seguente formato:

40 % A = # # # . #   B = # # . #

La gamma dei caratteri disponibili per definire il formato è variabile. Tuttavia i caratteri più soventemente utilizzati sono:

#	:	Indica una cifra, un carattere bianco o un segno –.
D	:	Stampa degli 0 se le cifre corrispondenti non esistono nel numero stampato.
.	:	Il punto indica il punto (virgola) decimale.
,	:	La virgola indica la stampa effettiva di una virgola.
$	:	Indica la stampa di un carattere $ all'inizio della zona associata a un numero. Se si specifica $$, l'$ sarà unito al numero.
+	:	Indica la stampa di un segno (+ o –).
–	:	Stampa un segno – se il numero è negativo, o un carattere bianco se il numero è positivo.
*	:	Stampa un carattere * se non ci sono delle cifre in questa posizione.

Nota. – Questo elenco di caratteri può leggermente variare da un sistema all'altro. È dunque importante consultare i cataloghi del sistema utilizzato per un uso corretto di questa istruzione.

Il vantaggio di questa istruzione sta nel fatto che facilita la stampa dei risultati.

LA MESSA A PUNTO DEI PROGRAMMI

Nel BASIC interpretato, la messa a punto dei programmi è relativamente agevole, qualunque sia il sistema, in quanto l'esecuzione del programma è interattiva.

Spesso non c'è bisogno di programmi specifici d'aiuto per la messa a punto (programma di «debugging»).

In effetti, se vi sono errori di sintassi, questi sono rilevati durante l'interpretazione dell'istruzione corrispondente. Se si tratta di un errore di esecuzione, il messaggio di errore indica l'istruzione in causa.

Tuttavia, bisogna notare che, se non si comprende la causa di un errore, durante l'esecuzione, può essere necessario inserire degli ordini di uscita (PRINT) dei risultati intermedi, proprio prima dell'istruzione che provoca l'errore. Si elimineranno queste istruzioni quando l'errore sarà stato rilevato.

In alcuni sistemi esistono delle possibilità di messa a punto grazie a comandi del tipo «TRACE», che permettono di ottenere la stampa o la visualizzazione delle istruzioni di programma in corso di esecuzione.

È possibile anche inserire dei punti di arresto nel programma («break-points») che permettono di verificare la parte già svolta e di riprendere eventualmente l'esecuzione passo per passo, cioè istruzione per istruzione.

Queste facilitazioni sono fornite solo su sistemi professionali e, in particolare, sono necessarie per le versioni di BASIC compilate. In effetti, quando un programma è giudicato corretto sintatticamente dal programma compilatore, può essere necessario usare tali strumenti, per mettere a punto il programma stesso.

Tuttavia, nelle versioni interpretate del BASIC, questi strumenti sono molto meno utili, in quanto è sempre possibile inserire delle istruzioni di uscita. Si vede allora l'utilità di numerare le istruzioni del programma per esempio di 10 in 10, poichè ciò permette di inserire o di cancellare delle istruzioni di uscita durante la messa a punto.

Prima di terminare questo paragrafo, è importante far notare a chi programma che non deve prendere l'abitudine di lanciarsi nella scrittura di un programma, prima di averlo ben analizzato sulla carta. Infatti, ed è uno dei rischi dei programmi interpretati, una cattiva analisi porta sempre a pessimi programmi.

Infatti, se ci si accorge che una prima versione non funziona, si può certo tentare di correggerla direttamente, ma, se il numero di correzioni è troppo elevato, è meglio riprendere il programma, o la parte di programma, incriminato e scriverlo di nuovo sulla carta, al fine di ottenere una versione definitiva chiara e ben strutturata. Ciò è importante specialmente se il programma è destinato ad essere utilizzato o eventualmente modificato in seguito.

I principali tipi di errore e la loro interpretazione

Si tratta di un problema importante che sconcerta e scoraggia il principiante quando non comprende i suoi errori. La comprensione non è molto facilitata a causa della laconicità dei messaggi d'errore e a causa della loro inadeguatezza a precisare ciò che è stato possibile svolgere a livello di linguaggio evoluto e non a livello di sistema. Infine, certamente, la lingua utilizzata (molto spesso questi messaggi sono in inglese) può porre dei problemi di comprensione e di interpretazione dei messaggi stessi.

Lo scopo di questa parte non è di dare una interpretazione dei messaggi d'errore nei diversi sistemi, ma è di aiutare l'utente a prevenire gli errori più frequenti e di mostrare il metodo di ricerca di un errore.

Gli errori di sintassi

Sono errori relativamente facili da trovare. Bisogna innanzitutto esaminare ciascun elemento dell'istruzione: il numero dell'istruzione (esiste già?), le parole chiave sono corrette ortograficamente ? i nomi delle variabili e delle costanti sono corretti ? i separatori (bianco, punto, virgola, apostrofo, ecc.) sono presenti e piazzati correttamente ?

Successivamente, se si tratta di una istruzione complessa, si dovrà studiare la sua struttura. Per le istruzioni aritmetiche, non si sono dimenticati parentesi, operatori o separatori ? esistono miscugli con variabili non numeriche ? Per le istruzioni di test, bisogna innanzitutto verificare l'istruzione condizionale: gli operatori relazionali o logici sono ben piazzati ? Successivamente si esaminerà l'istruzione che segue THEN. Per le istruzioni FOR, bisogna verificare le espressioni utilizzate per i valori iniziali e terminali e il passo; bisogna anche verificare che l'istruzione NEXT sia associata ad un adeguato parametro di controllo.

Infine, per le altre istruzioni, si verificherà che le etichette delle istruzioni GO TO e GO SUB siano ben scritte e che i separatori nelle istruzioni di entrata-uscita siano ben piazzate. Bisogna poi verificare che le variabili indicizzate siano dichiarate con le loro dimensioni.

Gli errori di esecuzione

Si distinguono due casi: quello in cui il programma ha già girato e il caso contrario.

Se il programma non ha mai girato, può trattarsi di un errore di logica a livello dell'analisi o della scrittura del programma. Se non dà alcun risultato, si dovrà verificare che il programma non si arresti. Se il programma dà risultati falsi, si dovrà tentare di localizzare il punto in cui si produce l'errore.

Se si tratta di un calcolo, bisognerà verificare che le istruzioni aritmetiche siano corrette e specialmente che la gerarchia degli operatori sia rispettata.

Se il programma fa il contrario di quello che avrebbe dovuto fare, bisognerà verificare che, nelle istruzioni di test, le condizioni siano ben definite.

Bisogna poi verificare che non si tratti semplicemente di un errore al livello di entrata - uscita.

Nel caso in cui si utilizzino anelli FOR, bisogna verificare i limiti e soprattutto i problemi di intervallo: un giro di troppo o un giro di meno può dare dei risultati aberranti.

Se tutto ciò è corretto, si dovrà far ricorso a inserzioni d'ordine PRINT, per determinare in modo preciso il gruppo di istruzioni a partire dalle quali i risultati diventano falsi.

Caso in cui il programma ha già funzionato

Nel caso in cui il programma ha già funzionato, gli errori che si possono verificare dopo diverse esecuzioni sono dovuti essenzialmente al fatto che la gamma di dati varia da una esecuzione all'altra. Possono presentarsi due casi. Il primo caso corrisponde spesso a dati particolari che hanno dato il controllo, a parti di programma che non sono ancora state utilizzate, rivelando così errori di programmazione non ancora scoperti. In alcuni casi, non è stata prevista la possibilità di dati particolari, e ciò provoca degli errori di esecuzione: *supero di capacità* (*Overflow*), divisioni per 0..., ecc. È dunque importante prevedere una gamma di dati tali da mettere in conto tutti i casi particolari. Il secondo caso corrisponde a funzionamenti aberranti del programma, dovuti a dati errati o a cattive manipolazioni da parte dell'utente. Questo tipo di errori è spesso inatteso in quanto non ci si aspetta di fare errori del genere. Nondimeno, questo tipo di errore verifica la robustezza del programma. In effetti, qualunque sia il problema da risolvere, se il programma è destinato ad essere utilizzato da persone diverse da coloro che l'hanno scritto, è estremamente importante prevedere «parapetti» e messaggi d'errore per tutte le cattive manipolazioni e per tutti i dati erronei. Del resto, si considera che il corpo principale del programma abbia un numero di istruzioni inferiore al numero di istruzioni destinate a trattare gli errori di questo tipo!

Questo lavoro è assai ingrato, ma è necessario per ottenere programmi «robusti».

Prima di far funzionare un programma, è consigliabile di darlo in mano a persone non sperimentate per verificare la sua robustezza. In questa materia, è sempre l'utente che ha ragione, e non serve a niente difendersi dicendo che questi ha fatto una cattiva manovra. Tutto il programma deve essere protetto al massimo contro tutte le manipolazioni erronee.

Infine, per terminare questo paragrafo, è ugualmente importante prevedere messaggi d'errore sufficientemente espliciti, in modo che l'utente possa capire l'errore commesso. In particolare, l'uso di test del tipo SE ERRORE VAI A (ONERR GO TO...) è raccomandato per tutti i programmi destinati ad utenti che non conoscono i messaggi abituali di un sistema BASIC.

CONCLUSIONE

In questo libro abbiamo presentato le differenti caratteristiche e possibilità del linguaggio BASIC. I primi quattro capitoli riguardano lo studio della struttura e delle istruzioni standard del linguaggio. Nel medesimo tempo, i principi e le tecniche di base della programmazione sono state presentate in modo progressivo per permettere al principiante la comprensione della programmazione in modo più facile e rapido.

Effettivamente, secondo tutte le statistiche, il linguaggio BASIC è il linguaggio che permette di realizzare più rapidamente programmi operazionali. La nostra esperienza di insegnamento conferma che è anche il linguaggio più rapidamente assimilato da utenti non professionisti dell'informatica (economisti, medici, contabili, segretari, ... ecc.). Ciò non vuol dire che il linguaggio BASIC sia il miglior linguaggio di programmazione: esistono linguaggi come il PASCAL che sono certo meglio strutturati e che danno programmi più facili da leggere e da conservare.

Rimane il fatto che il BASIC è un linguaggio in cui il sistema di sfruttamento della macchina diventa quasi trasparente per l'utente principiante.

Per il professionista o per l'insegnante, la disponibilità del BASIC su microelaboratori autonomi permette la scrittura o la messa a punto rapida dei programmi per verificare un algoritmo, ottenere rapidamente un risultato, visualizzare una curva, mettere in evidenza dei piccoli archivi, ecc.

Negli ultimi capitoli (5, 6 e 7) sono state presentate delle estensioni del linguaggio che non sono standardizzate. Abbiamo deliberatamente scelto di presentare le possibilità esistenti su piccole macchine. Qui la cosa importante è di aver afferrato i concetti di base: la nozione di primitiva ad un sistema di files, la generazione di vettori, di forme, i cambiamenti di origine e di scala nei metodi grafici.

Sicuramente un sistema potrà realizzare questi concetti in maniera più flessibile di un altro, e gli esempi dati avevano come scopo quello di mostrare cosa era possibile fare con un sistema minimale.

Contrariamente a ciò che si pensa spesso, i sistemi microelaboratori detti «personali» non sono i meno servizievoli, nè i meno flessibili. Certamente, qualche volta mancano di «parapetti» e i sistemi di files sono spesso rudimentali, ma ciò evolve rapidamente, grazie alla possibilità di ammortizzare logiche più sofisticate su un gran numero di unità.

Speriamo dunque che questa opera abbia risposto all'attenzione di tutti gli utenti principianti e che permetterà loro non solo di dominare il linguaggio BASIC, ma di sviluppare proprie applicazioni nonchè di immaginarne di nuove e più complesse.

Infine, noi pensiamo che, per coloro che considerano il linguaggio BASIC solo come inizio, malgrado le imperfezioni di questo, l'apprendimento di un linguaggio evoluto costituisca un trampolino utilissimo per dominare rapidamente tutti gli altri linguaggi di programmazione.

APPENDICE 1

RICHIAMI DI NUMERAZIONE BINARIA

Il linguaggio binario

Definizione: Il linguaggio binario è un linguaggio in cui l'alfabeto è composto da due caratteri:

$$A = \{0, 1\}$$

- Le parole di questo linguaggio sono dunque 0, 1, 10, 11, 100, 101, 110, 111...

Può essere utilizzato come sistema di codifica di un insieme.

Esempio:

$$0 \leftrightarrow a_1$$
$$1 \leftrightarrow a_2$$
$$10 \leftrightarrow a_3$$

Sistema di numerazione binaria

Il linguaggio binario può anche essere utilizzato come *sistema di numerazione* che associa a ciascun carattere binario un valore corrispondente a una potenza di 2.

Così, alla posizione più a destra, si assocerà la potenza 0, alla posizione seguente, andando verso sinistra, la potenza 1, successivamente la potenza 2, e così via. Questo sistema si chiama sistema di numerazione per posizione.

Principio

Un numero N sarà rappresentato in base b da:

$$N = a_0 \times b^0 + a_1 \times b^1 + a_2 \times b^2 + ... + a_n \times b^n$$

con:

$$0 \leq a_i < b$$

Casi particolari

— se $b = 10$:

$$N = a_0 \times 1 + a_1 \times 10 + a_2 \times 10^2 + \dots a_n \times 10^n$$

$$0 \leq a_1 < 10$$

dove a_0 rappresenta la cifra delle unità, a_1 quella delle decine, a_2 quella delle centinaia...

Gli a_i corrispondono a un carattere decimale.

— Se $b = 2$:

$$N = a_0 \times 1 + a_1 \times 2 + a_2 \times 2^2 + \dots a_n \times 2^n$$

Gli a_i corrispondono a un carattere binario.

Esempi:

— *Il numero decimale 1789 è dato da:*
$9 \times 1 + 8 \times 10 + 7 \times 10^2 + 1 \times 10^3 = 9 + 80 + 700 + 1000$
— *Il numero binario 1101 significa:*
$1 \times 1 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = 1 + 0 + 4 + 8 = 13_{10}$

Questo numero rappresenta il numero intero 13 in decimale.

Nota. — Considerare il carattere più a destra come la cifra di peso più basso è solo una convenzione, e, nella misura in cui il numero viene scritto da sinistra verso destra, ciò può sembrare perlomeno strano. Bisogna ricordare però che il sistema di numerazione per posizione ci deriva dai matematici arabi, i quali scrivono da destra verso sinistra.

Quando si considera il sistema di numerazione binario, una cifra binaria è chiamata «*bit*» (contrazione delle parole inglesi «binary digit»).

Proprietà

Dopo quanto abbiamo visto, una parola binaria di n bits permette di rappresentare i numeri interi seguenti:

$$0 \leq N \leq 2^n - 1$$

Esempio:

— Una parola di otto bits permette di rappresentare i numeri interi tra 0 e $2^8 - 1 = 255$.

Numero di bits necessari per la rappresentazione di un numero intero

Inversamente, se si vuole rappresentare un numero N in binario, sarà necessario un numero di $n = \lceil \mathrm{Log}_2 N \rceil$ bits, dove il simbolo $\lceil \quad \rceil$ indica l'intero immediatamente superiore al valore calcolato.

In pratica, non disponendo di una tavola dei logaritmi, si può determinare questo numero *n*, sviluppando il numero N con potenze del 2.

$$2^{n-1} \leq N \leq 2^{n}$$

In questo caso, si sa che, per rappresentare N, sono necessari *n* bits.

Esempio:

Se N = 746, si ha:

$$n = \lceil \mathrm{Log}_2 N \rceil = 10$$

In effetti:

$$2^9 = 512 < 746 < 2^{10} = 1024$$

Conversione binaria decimale

La conversione di un numero binario in un numero decimale deriva dal principio di numerazione.

L'algoritmo di conversione può riassumersi così:

1. Incominciare dalla destra del numero;
2. Portare N a 0 e *n* a 0;
3. Leggere il carattere seguente verso sinistra;
4. Se il carattere è un bianco, allora ci si ferma;
5. Se il carattere è uno 0, *allora* fare $n + 1$ e continuare in 3. *Altrimenti* calcolare $N + 2^n$ e continuare in 3.

Esempio:

– 1 0 1 1 1 0 rappresenta in decimale il numero
  N = 2 + 4 + 8 + 32 = 46

Nota. • un numero pari termina con un bit 0;
• un numero dispari termina con un bit 1.

Conversione decimale binaria

Se si considera la divisione per 2 di un numero intero N, si ha:

$$N = 2 \times q_0 + r_0 \qquad 0 \leq r_0 \leq 1$$

Se $q_0 > 2$, si ha:

$$q_0 = 2\,q_1 + r_1 \qquad 0 \leq r_1 \leq 1$$

Sia $N = 2^2\,q_1 + r_1 + r_0$

Se $q_1 > 2$:

$$q_1 = 2\,q_2 + r_2 \qquad 0 \leq r_2 \leq 1$$

Sia:

$$N = 2^3\,q_2 + 2^2 \times r_2 + 2 \times r_1 + r_0$$

Se si continuano allo stesso modo le divisioni per 2 fin quando è possibile, otterremo:

$$N = 2^i q_i + 2^{i-1}\,r_{i-1} + ... + 2^3\,r_3 + 2^2\,r_2 + 2\,r_1 + r_0$$

con $0 \leq q_i \leq 1$.

Ciò mostra che i resti delle divisioni successive per 2 rappresentano, per definizione, le cifre binarie associate al numero N.

Esempio:

Si debba convertire N = 59. Le divisioni successive sono:

59	2				
1	29	2			
	1	14	2		
		0	7	2	
			1	3	2
				1	1

In binario avremo: 111011 (si leggono i resti dal basso verso l'alto).

La rappresentazione ottale ed esadecimale

Praticamente i numeri binari sono difficili da manipolare, a causa della loro lunghezza e monotonia (serie di 0 e 1).

Per condensare questi numeri si utilizzano rappresentazioni più compatte.

Rappresentazione ottale

Si tratta di un sistema di numerazione a base 8, che ha per alfabeto:

$$A = \{ 0, 1, 2, 3, 4, 5, 6, 7 \}$$

Il passaggio da binario in ottale è immediato, considerando i gruppi di tre bits successivi di un numero binario.

000	0	100	4
001	1	101	5
010	2	110	6
011	3	111	7

Così, il numero binario 101110, in ottale, si scrive 56, poichè 101 = 5 e 110 = 6.

Ugualmente immediata è la conversione inversa: ciascuna cifra ottale è sostituita dal suo equivalente binario.

Rappresentazione esadecimale

Si tratta di un sistema di numerazione a base 16, avente per alfabeto:

0, 1, 2, ... 9, A, B, C, D, E, F

In decimale	10	A	1010 in binario
	11	B	1011
	12	C	1100
	13	D	1101
	14	E	1110
	15	F	1111

Il passaggio da binario in esadecimale si attua considerando gruppi di quattro bits successivi e sostituendoli con una cifra esadecimale.

Ad esempio, si abbia il numero binario 10111001.

Questi viene tradotto in esadecimale con B9, poichè:

1011 = B
1001 = 9

La conversione in senso inverso è analoga: ciascuna cifra esadecimale corrisponde ad una serie di quattro bits.

Il codice binario decimale (BCD)

Alcuni elaboratori, pur trattando in modo interno parole di linguaggio binario, utilizzano, per la rappresentazione dei numeri decimali, un codice che associa a ciascuna cifra decimale la sua rappresentazione binaria.

Sia:	0	0
	1	1
	2	10
	3	11
	4	100
	5	101
	6	110
	7	111
	8	1000
	9	1001

In questo caso, la rappresentazione dei numeri decimali *non utilizza* il

principio di numerazione binaria aldilà della cifra 9, ma ciascuna cifra di un numero è rappresentata dal suo codice.

Esempi:

- 78 è rappresentato da: 01111000
- 99 è rappresentato da: 10011001

Ciò implica che tutte le configurazioni binarie di una parola non siano decifrabili in BCD.

In pratica, tutte le configurazioni di quattro bits che vanno aldilà di 1001 sono indecifrabili.

In questo modo le serie di quattro bits seguenti sono scorrette: 1010, 1011, 1101, 1110, 1111.

Addizione e moltiplicazione in binario

1. Addizione: il principio dell'addizione binaria è analogo a quello dell'addizione decimale.

La tabella dell'addizione binaria si limita a quattro combinazioni di digits possibili:

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 10$$

Il solo caso in cui si ha un riporto è l'ultimo. Il digit corrispondente al risultato è 0 e il riporto di 1 viene inserito nel digit seguente.

Esempio:

```
   11
   1011        11
 + 1101      + 13
 ------      ----
  11000        24
```

Nota. — Nel caso in cui la dimensione delle parole è fissata in n, il massimo numero ottenibile per numeri strettamente positivi è $2^n - 1$. Se si sommano due numeri la cui somma supera $2^n - 1$, si ha uno straripamento di capacità.

2. Moltiplicazione: la tabella della moltiplicazione binaria è limitata a:

$$0 \times 0 = 0$$
$$0 \times 1 = 0$$
$$1 \times 0 = 0$$
$$1 \times 1 = 1$$

Il solo caso in cui si ha un risultato non nullo è l'ultimo.

Algoritmo della moltiplicazione

1 — Mettere N = 0.
2 — Mettere il risultato P = 0.
3 — Leggere il carattere seguente verso la sinistra del moltiplicatore.
4 — Se questo carattere è bianco: stop.
5 — Se il carattere è 0, fare N + 1 e continuare in 3.
6 — Altrimenti aggiungere il moltiplicando seguito da N zeri a P, fare N + 1 e continuare in 3.

Esempio:

```
   1011
×   101
-------
   1011
 1011
-------
 110111
```

Nota. — La moltiplicazione di due parole di n bits, necessita di una parola di 2 n bits per il risultato.

Codifica dei numeri negativi

Il problema della rappresentazione dei numeri interi negativi presuppone una codifica del *segno.*

Ciò necessita di un bit: si può, ad esempio, prendere il bit più a sinistra di una parola di n bits e considerare che il suo significato sia:
+ se il bit è 0 — se il bit è 1.

In questo caso, si suppone che gli $n - 1$ bits restanti vengano utilizzati per rappresentare il valore assoluto.

Esempio:

La parola 00000101 rappresenta + 5
e la parola 10000101 rappresenta — 5

Questo sistema di codifica presenta diversi inconvenienti:
- per effettuare una operazione su interi positivi e negativi, bisogna valutare sistematicamente il bit del segno;
- bisogna definire una operazione di sottrazione dei valori assoluti;
- infine, la configurazione 10000000 può essere interpretata come uno zero negativo.

In questo sistema si hanno dunque due codici per rappresentare lo zero. E ciò non è affatto soddisfacente.

La rappresentazione in complemento a 1

Questa rappresentazione utilizza come convenzione di rappresentazione di un numero negativo il complemento a 1 di ciascun bit del numero che ha lo stesso valore assoluto, ma segno positivo.
Il complemento a 1 di 0 è 1
Il complemento a 1 di 1 è 0

Esempio:

+ 6	0110
– 6	1001

Questa convenzione utilizza ugualmente il bit più a sinistra per rappresentare il segno.
L'operazione di negazione è semplice e identica su tutti i bits. L'addizione bit a bit di un numero e del suo opposto dà 1111, che è il complemento a 1 di 0000.
In questa configurazione si ha ancora l'inconveniente del doppio zero (+ 0 e – 0).

Convenzione in complemento a 2

Se si considera la sottrazione di due cifre decimali $9 - 3 = 6$, e si considera il complemento a 10 di 3, questo è 7.
Se si fa la somma $9 + 7$, si ottiene 16, cioè la cifra 6 associata al riporto di 1.
Ignorando questo riporto, la sottrazione di 3 da 9 è dunque equivalente all'addizione senza riporto di 9 con il complemento a 10 di 3, cioè 7 (questa è una teoria del tutto generale ed è lo stesso principio utilizzato in binario per il complemento a 2).
Per ottenere il complemento a 2 di un numero binario, è sufficiente prenderne il complemento a 1 e aggiungere 1 al risultato.

Esempio:

Si abbia il numero binario 00001011.
Il suo complemento a 1 (ottenuto prendendo l'opposto di ciascun bit) è 11110100.
Il suo complemento a 2 si ottiene aggiungendo 1 al complemento a 1, e cioè 11110101.
L'addizione di un numero e del suo opposto dà:

$$\begin{array}{r} 00001011 \\ \underline{11110101} \\ 100000000 \end{array}$$

Si vede dunque che questa volta il numero ottenuto su otto bits è 00000000, mentre il riporto si è in effetti inserito nel nono bit, che non appartiene alla parola di otto bits.

Questa volta si ha dunque una rappresentazione tale per cui:

$$a + (-a) = 0$$

D'altra parte, in questo modo di rappresentazione, il numero binario 10000000 è, per definizione, uguale a − 128: in effetti, si tratta di un numero negativo, il cui valore assoluto è:

$$10000000, \text{ cioè } 2^7 = 128$$

La convenzione complemento a due possiede dunque un solo zero.

Generalizzazione

La convenzione complemento a 2 è attualmente generalizzata in tutti gli elaboratori.

Se si ha una parola di n bits, i numeri che si possono rappresentare in convenzione complemento a due sono tali che:

$$-2^{n-1} \leq N \leq 2^{n-1} - 1$$

Esempio:

Su una macchina che lavora su parole di sedici bits, si ha

$$-2^{15} \leq N \leq 2^{15} - 1 \qquad \text{e cioè} \qquad -32768 \leq N \leq 32767$$

La rappresentazione di numeri reali

Volendo poter trattare numeri frazionari, è necessario definire un nuovo modo di rappresentazione dei numeri.

Rappresentazione in virgola fissa

Una prima soluzione è di considerare una parte intera e una parte decimale separate da una virgola la cui posizione sarà fissata a priori.

Si abbia una parola di otto bits: possiamo definire la posizione della virgola a metà della parola, in modo da avere quattro bits per la parte intera e quattro per la parte decimale.

$$N = a_3 \times 2^3 + a_2 \times 2^2 + a_1 \times 2 + a_0$$
$$+ a_{-1} \times 2^{-1} + a_{-2} \times 2^{-2} + a_{-3} \times 2^{-3} + a_{-4} \times 2^{-4}$$

In questo caso, si possono rappresentare numeri positivi frazionari che vanno da 0000.0000 a 1111.1111 e cioè da 0.0 a 15.9375.

Il campo di variazione, come si vede, è estremamente ridotto.

Se, per di più, si considerano i numeri positivi e negativi, si potrà variare solo da − 8.9375 a + 7.9375.

La rappresentazione in virgola fissa è dunque assai limitata per il numero di bits di cui si dispone.

Rappresentazione in virgola fluttuante

Un'altra soluzione è di considerare un numero frazionario con l'aiuto della rappresentazione esponenziale (logaritmica).

Per esempio, si abbia il numero:

$$1984.128$$

Può essere rappresentato sotto differenti forme equivalenti:

$$19.84128 \times 10^{2}$$
$$0.1984128 \times 10^{4}$$
$$1984128 \times 10^{-3}$$

Si vede dunque che, facendo variare il valore della potenza, è possibile variare la posizione della virgola; da ciò deriva il nome della virgola fluttuante.

In binario, questa rappresentazione corrisponde a un numero binario moltiplicato per una potenza di 2. Ad esempio:

$$11011.1101$$
$$= 110111101 \times 2^{-4}$$
$$= 0.110111101 \times 2^{5}$$

Rappresentazione normalizzata

La rappresentazione normalizzata è caratterizzata dalla posizione della virgola (del punto) tale per cui la prima cifra significativa sia a destra della virgola.

Esempio:

0.110111101×2^{5} è una rappresentazione normalizzata.

Definizione: Si chiama *mantissa* l'insieme delle cifre significative (bits) in rappresentazione normalizzata.

L'esponente è la potenza di 2 associata a questa rappresentazione normalizzata.

Definizione: Un numero fluttuante è caratterizzato da un numero binario (positivo o negativo) rappresentante la mantissa, associato ad un numero binario (positivo o negativo) che rappresenta l'esponente o caratteristica.

Esempio:

Si può ad esempio definire un numero fluttuante con il seguente schema:

S	ESPONENTE	S	MANTISSA
	8 bits		24 bits

Potenze di 2

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 164	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 062 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 676 950 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.000 000 000 000 000 000 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0.000 000 000 000 000 000 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0.000 000 000 000 000 000 108 420 217 248 550 443 400 745 280 086 994 171 142 578 125

Potenze di 16 (in base 10)

16^n	n	16^{-n}				
1	0	0.10000	00000	00000	00000	$\times 10$
16	1	0.62500	00000	00000	00000	$\times 10^{-1}$
256	2	0.39062	50000	00000	00000	$\times 10^{-2}$
4 096	3	0.24414	06250	00000	00000	$\times 10^{-3}$
65 536	4	0.15258	78906	25000	00000	$\times 10^{-4}$
1 048 576	5	0.95367	43164	06250	00000	$\times 10^{-6}$
16 777 216	6	0.59604	64477	53906	25000	$\times 10^{-7}$
268 435 456	7	0.37252	90298	46191	40627	$\times 10^{-8}$
4 294 967 296	8	0.23283	06436	53869	62891	$\times 10^{-9}$
68 719 476 736	9	0.14551	91522	83668	51807	$\times 10^{-10}$
1 099 511 627 776	10	0.90949	47017	72928	23792	$\times 10^{-12}$
17 592 186 044 416	11	0.56843	41886	08080	14870	$\times 10^{-13}$
281 474 976 710 656	12	0.35527	13678	80050	09294	$\times 10^{-14}$
4 503 599 627 370 496	13	0.22204	46049	25031	30808	$\times 10^{-15}$
72 057 594 037 927 936	14	0.13877	78780	78144	56755	$\times 10^{-16}$
1 152 921 504 606 846 976	15	0.86736	17379	88403	54721	$\times 10^{-18}$

Potenze di 10 (in base 16)

			10^n	n			10^{-n}		
			1	0	1.0000	0000	0000	0000	
			A	1	0.1999	9999	9999	999A	
			64	2	0.28F5	C28F	5C28	F5C3	$\times 16^{-1}$
			3E8	3	0.4189	374B	C6A7	EF9E	$\times 16^{-2}$
			2710	4	0.68DB	8BAC	710C	B296	$\times 16^{-3}$
		1	86A0	5	0.A7C5	AC47	1B47	8423	$\times 16^{-4}$
		F	4240	6	0.10C6	F7A0	B5ED	8D37	$\times 16^{-4}$
		98	9680	7	0.1AD7	F29A	BCAF	4858	$\times 16^{-5}$
		5F5	E100	8	0.2AF3	1DC4	6118	73BF	$\times 16^{-6}$
		3B9A	CA00	9	0.44B8	2FA0	9B5A	52CC	$\times 16^{-7}$
	2	540B	E400	10	0.6DF3	7F67	SEF6	EADF	$\times 16^{-8}$
	17	4876	E800	11	0.AFEB	FF0B	CB24	AAFF	$\times 16^{-9}$
	E8	D4A5	1000	12	0.1197	9981	2DEA	1119	$\times 16^{-9}$
	918	4E72	A000	13	0.1C25	C268	4976	81C2	$\times 16^{-10}$
	5AF3	107A	4000	14	0.2D09	370D	4257	3604	$\times 16^{-11}$
3	8D7E	A4C6	8000	15	0.480E	BE7B	9D58	566D	$\times 16^{-12}$
23	8652	6FC1	0000	16	0.734A	CA5F	6226	F0AE	$\times 16^{-13}$
163	4578	5D8A	0000	17	0.B877	AA32	36A4	B449	$\times 16^{-14}$
DE0	B6B3	A764	0000	18	0.1272	5DD1	D243	ABA1	$\times 16^{-14}$
8AC7	2304	89E8	0000	19	0.1D83	C94F	B6D2	AC35	$\times 16^{-15}$

Rappresentazione dei caratteri

La rappresentazione dei caratteri in macchina viene attuata attraverso un codice.

Volendo rappresentare l'insieme delle lettere dell'alfabeto e le cifre, cioè un totale di 36 caratteri, abbiamo bisogno di 6 bits, infatti:

$$32 < 36 < 64 = 2^6$$

Aggiungendo le minuscole e i caratteri speciali, saranno necessari 7 bits.
Storicamente, innanzitutto sono stati utilizzati codici di 6 bits. Successivamente, si sono imposti i codici con 7 bits (codifica ASCII). Tuttavia, alcuni costruttori utilizzano un codice di 8 bits (codifica EBCDIC).

Attualmente, in pratica, vengono utilizzati codici di 8 bits (ASCII o EBCDIC).

Diamo qui sotto le tabelle di questi codici.

TABELLA DEI CODICI ASCII

Caratteri o controllo	ASCII o esadecimali	Caratteri speciali e cifre	ASCII o esadecimali	Caratteri maiuscoli	ASCII o esadecimali	Caratteri minuscoli	ASCII o esadecimali
NUL	00	SP (spazio)	20	@	40	\	60
SOH	01	!	21	A	41	a	61
STX	02	"	22	B	42	b	62
ETX	03	#	23	C	43	c	63
EOT	04	$	24	D	44	d	64
ENQ	05	%	25	E	45	e	65
ACK	06	&	26	F	46	f	66
BEL	07	/	27	G	47	g	67
BS	08	(	28	H	48	h	68
HT	09	)	29	I	49	i	69
LF	0A	*	2A	J	4A	j	6A
VT	0B	+	2B	K	4B	k	6B
FF	0C	,	2C	L	4C	l	6C
CR	0D	–	2D	M	4D	m	6D
SO	0E	.	2E	N	4E	n	6E
SI	0F	/	2F	O	4F	o	6F
DLE	10	0	30	P	50	p	70
DCA (X-ON)	11	1	31	Q	51	q	71
DC2 (TAPE)	12	2	32	R	52	r	72
DC3 (X-OFF)	13	3	33	S	53	s	73
DC4 (TAPE)	14	4	34	T	54	t	74
NAK	15	5	35	U	55	u	75
SYN	16	6	36	V	56	v	76
ETB	17	7	37	W	57	w	77
CAN	18	8	38	X	58	x	78
EM	19	9	39	Y	59	y	79
SUB	1A	:	3A	Z	5A	z	7A
ESC	1B	;	3B	[	5B	{	7B
FS	1C	<	3C	\	5C	¦	7C
GS	1D	=	3D	]	5D	}(ALT MODE)	7D
RS	1E	>	3E	Δ(↑)	5E		7E
US	1F	?	3F	– (←)	5F	DEL	7F

APPENDICE 2

DEFINIZIONI SINTATTICHE DEL LINGUAGGIO BASIC

La sintassi di un linguaggio viene definita con l'aiuto di un «meta linguaggio» che precisa gli oggetti del linguaggio e le loro relazioni, cioè le regole di costruzione delle parole e delle frasi del linguaggio. Il metalinguaggio più utilizzato per questo scopo è quello definito come «la forma normale di BACKUS NAUR» (BNF).

I metasimboli utilizzati sono i caratteri:
: = , che indicano una definizione;
il carattere I , che indica una alternativa, quando esistono più possibilità.

Si utilizzano poi i caratteri < > per indicare un meta-oggetto del linguaggio definito da una successione.

Esempio:

< Cifra > : = 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 / 0

Se un meta-oggetto è opzionale, lo si indica con parentesi quadre [ ].
Se un meta-oggetto è ripetuto spesso, lo si indica utilizzando il carattere *.

Esempio:

<numero> : = <cifra> *

Il carattere bianco è indicato così: b̸.

Si hanno allora le seguenti definizioni:

<Carattere> : = <lettera> I <cifra> I <carattere speciale>

<Lettera> : = A / B / C / D / E / F / G / H / I / J / K / L / M / N / O / P / Q / R / S / T / U / V / W / X / Y / Z

<cifra> : = 0 / 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8 / 9 /

<Carattere speciale> : = ƀ / ’ / ” / (/) / : / ; / . / $ / % /   / ! / ? / & / * /
+ / − / = / / 7 / > / <

<Nome di variabile> : <lettera> [<lettera | cifra> *]
(in pratica, nella maggior parte dei BASIC, ci si limita a due caratteri alfanumerici)

<Nome della variabile intera> : = <nome della variabile> %

<Nome della variabile stringa di caratteri> : = <nome della variabile> $

<Intero> : = [+ | −] <cifra> *

<Reale> : = [+ | −] <cifra> * [<cifra> *]
[E [+ | −] <cifra> *]

<Stringa di caratteri> : = “ [<carattere>] ”

<Linea di istruzione> : = <n° di istruzioni> [<istruzioni :> *]
<istruzioni> ®
(® simbolizza il ritorno alla linea)

<N° di istruzioni> : = <cifra> <cifra> *

<Istruzione> : = <Nota> | <istruzione di dichiarazione> | <istruzione eseguibile>.

<Nota> : = REM [<carattere> *]

<Istruzione di dichiarazione> : = <definizione di dati>
<definizione di funzione>
<definizione di elenco o tabella>

<Definizione di dato> : = DATA [<dati> , *] <dati>

<Dato> : = <intero> | <reale> | <stringa di caratteri>

<Definizione di elenco o tabella> : = DIMENSION [<variabile> <dimensione>] * <variabile> <dimension>

<Variabile> : = <nome della variabile> | <nome della variabile intera> |
<nome della variabile stringa di caratteri>

<Dimensione> : = (<variabile numerica> [, <variabile numerica>] *)

<Variabile numerica> : = <intero> | <nome della variabile> | <nome della variabile intera>

<Definizione di funzione> : = DEF FN <nome della funzione> = <espressione aritmetica>

<Nome della funzione> : = <lettera> [<lettera>]

<Istruzione eseguibile> : = <istruzione di assegnazione> | <istruzione di test> | <istruzione di iterazione> | <istruzione di entrata-uscita> | <istruzione di salto> | <istruzione di scambio> | <istruzione fine> | <istruzione d'arresto> |

<Istruzione fine> : = END

<Istruzione d'arresto> : = STOP

<Istruzione di salto> : = <Parola di salto> <n° di istruzione>

<Parola di salto> : = GO TO / GO SUB

<Istruzione di scambio> : = ON <espressione aritmetica> <parola di salto> <n° di istruzione> | <n° di istruzione>] *

<Istruzione di entrata-uscita> : = <istruzione di entrata> | <istruzione di uscita>

<Istruzione di entrata> : = <istruzione di lettura dei dati definita nel programma> | <istruzione di entrata esterna>

<Istruzione di lettura di dati interni> : = READ <variabile> | , <variabile> | *

<Istruzione di entrata esterna> : = INPUT | <stringa di caratteri> ; | <variabile> | , <variabile> | * | GET <variabile>

<Istruzioni di uscita> : = PRINT <espressione> | , | ; <espressione> | *

<Istruzioni di assegnazione> : = <istruzione aritmetica> | <istruzione di manipolazione di stringa di caratteri>

<Istruzione aritmetica> : = [LET] <variabile> | (<indice>) | = <espressione aritmetica>

<Espressione aritmetica> : = [+ | –] <termine> | <espressione aritmetica> <op di addizione> <termine>

<Termine> : = <fattore> | <termine> <operaz. di moltiplicazione> <fattore>

<Op. di addizione> : = + | –

<Op. di moltiplicazione> : = * | /

<Fattore> : = <variabile numerica> | <costante numerica> | <funzione>] (<espressione aritmetica>)] <espressione aritmetica> ↑ <espressione aritmetica>

<Funzione> : = <nome della funzione> (<fattore>)

<Istruzione di manipolazione di caratteri> : =

<Variabile stringa> = <espressione stringa di caratteri>

<Espressione stringa di caratteri> : = <variabile stringa> | <espressione stringa> <operatore di concatenamento> <variabile stringa>

<Operatore di concatenamento> : = +

<Istruzioni di test> : = IF <condizione> THEN <istruzione> |

<Condizione> : = <espressione relazionale> | <espressione logica> | <espressione aritmetica | espressione stringa> <operatore relazionale> <espressione aritmetica | espressione stringa>

<Operatore relazionale> : = < | > | > = | < = | = | < >

<Espressione logica> : = NOT <espressione relazionale> | <espressione logica> <operatore logico> <espressione logica>] <espressione relazionale> <operatore logico> <espressione relazionale>

<Operatore logico> : = AND | OR

<Istruzione di iterazione> : = FOR <istruzione aritmetica> TO <espressione aritmetica> STEP <espressione aritmetica> <istruzione> * NEXT <variabile numerica>